80C196Mx
Demo Board
User's Manual

March 1997

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

CONTENTS

FIGURES

TABLES

1

Guide to This Manual

CHAPTER 1
GUIDE TO THIS MANUAL

This manual describes the use of the 80C196Mx Demo Board kit for developing and evaluating an embedded application design based on the 80C196 MCS® 96 microcontroller. This manual is intended for design engineers who are already familiar with the principles of microcontrollers.

1.1 MANUAL CONTENTS

This manual has six chapters and two appendices:

Chapter 2, "Getting Started with the 80C196Mx Demo Board" — includes a list of the kit contents and instructions on initializing the demo board and installing the software.

Chapter 3, "Introduction to the 80C196Mx Demo Board" — describes the 80C196Mx demo board; it includes a component-level diagram and describes the installation of memory devices.

Chapter 4, "Introduction to the Embedded Controller Monitor (ECM)" — introduces the user interface software, which comprises ECM96Mx and RISMMx.

Chapter 5, "ECM96Mx Commands" — describes the part of the Embedded Controller Monitor (ECM) that executes on the host PC.

Chapter 6, "RISM Registers and Commands" — describes the commands for the 80C196Mx reduced instruction set monitor (RISMMx), the part of the Embedded Controller Monitor (ECM) that executes on the demo board microcontroller.

Appendix A, "Components, Jumpers, and Connectors" — provides figures and tables to help you configure the 80C196Mx demo board and other information for you to consider as you develop an application.

Appendix B, "Parts List" — contains a listing of all discrete and active components for the 80C196Mx demo board.

1.2 NOTATIONAL CONVENTIONS AND TERMINOLOGY

The following notations and terminology are used throughout this manual.

— The pound symbol (#) has two meanings, depending on the context. When used with a signal name, the symbol means that the signal is active low. When used in an instruction, the symbol prefixes an immediate value in immediate addressing mode.

italics — Italics identify variables and introduce new terminology. The context in which italics are used distinguishes between the two possible meanings.

Variables in registers and signal names are commonly represented by *x* and *y*, where *x* represents the first variable and *y* represents the second variable. For example, in register P*x*_MODE.*y*, *x* represents the variable that identifies the specific port, and *y* represents the register bit variable [7:0 or 15:0].

X — Uppercase X (no italics) represents an unknown value or a "don't care" state or condition. The value may be either binary or hexadecimal, depending upon the context. For example, the hexadecimal value FF2XAFH indicates that bits 11:8 are unknown; 10XX in binary context indicates that the two LSBs are unknown.

Board Components — The following abbreviations are used to represent discrete and active components.

C*x*	capacitor
D*x*	diode
DP*x*	LED bank
E*x*	jumper
JP*x*	connector
L*x*	inductor
P*x*	port
R*x*	resistor
RP*x*	resistor pack
S*x*	switch
U*x*	device socket (e.g., latch, buffer, memory, controller)

Assert and Deassert The terms *assert* and *deassert* refer to the act of making a signal active (enabled) and inactive (disabled), respectively. The active polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low (equal to or less than V_{OL}); to assert ALE is to drive it high (equal to or greater than V_{OH}); to deassert RD# is to drive it high; to deassert ALE is to drive it low.

Instructions Instruction mnemonics are shown in upper case to avoid confusion. You may use either upper case or lower case in your source code.

NC The term "NC" is an abbreviation for "no connection." It indicates that no connection is required.

Numbers Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character H. Decimal and binary numbers are represented by their customary notations. (That is, 255 is a decimal number and 11111111 is a binary number. In some cases, the letter B is appended to binary numbers for clarity.)

Units of Measure The following abbreviations are used to represent units of measure:

A	amps, amperes
mA	milliamps, milliamperes
Kbyte	kilobytes
KHz	kilohertz
KΩ	kilo-ohms
Mbyte	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
V	voltage, volts
VDC	voltage, direct current
VAC	voltage, alternating current
W	watts

µA	microamps, microamperes
µF	microfarads
µs	microseconds

Register Bits Bit locations are indexed by 7:0 (or 15:0), where bit 0 is the least-significant bit and 7 (or 15) is the most-significant bit. An individual bit is represented by the register name, followed by a period and the bit number. For example, WSR.7 is bit 7 of the window select register. In some discussions, bit names are used. For example, the name of WSR.7 is HLDEN.

Register Names Register names are shown in upper case. For example, TIMER2 is the timer 2 register; timer 2 is the timer. If a register name contains a lowercase character, it represents more than one register. For example, P*x*_REG represents four registers: P1_REG, P2_REG, P3_REG, and P4_REG.

Reserved Bits Certain bits are described as *reserved* bits. In illustrations, reserved bits are indicated with a dash (—). These bits are not used in this device, but they may be used in future implementations. To help ensure that a current software design is compatible with future implementations, reserved bits should be cleared (given a value of "0"), unless otherwise noted.

Set and Clear The terms *set* and *clear* refer to the value of a bit or the act of giving it a value. If a bit is *set*, its value is "1"; *setting* a bit gives it a "1" value. If a bit is *clear,* its value is "0"; *clearing* a bit gives it a "0" value.

Signal Names Signal names are shown in upper case. When several signals share a common name, an individual signal is represented by the signal name followed by a number. For example, the EPA signals are named EPA0, EPA1, EPA2, etc. Port pins are represented by the port abbreviation, a period, and the pin number (e.g., P1.0, P1.1). A pound symbol (#) appended to a signal name identifies an active-low signal.

Command Lines For command line input to software, such as MS-DOS* and ECM96Mx, this manual uses notation described in Section 5.1, "ECM96Mx Command Notation."

1.3 RELATED DOCUMENTS AND PRODUCTS

The following lists refer to documents and products that are useful in designing systems using an 80C196Mx embedded microcontroller. The documents are available through Intel Literature. (Order literature by calling the "FaxBack* Service" on page 1-6 or the phone numbers for "Product Literature" on page 1-8).

Embedded Microcontrollers	Order Number 270646
Embedded Applications	Order Number 270648
Development Tools	Order Number 272326
Packaging	Order Number 240800

1.3.1 Data Sheet and User's Manual Supplement

The data sheets are included in the *Embedded Microcontrollers* handbook and are also available individually.

8XC196MC, 8XC196MD, 8XC196MH Microcontroller User's Manual	Order Number 272181

1.3.2 Application Notes

These application notes are included in the *Embedded Applications* handbook and are also available individually.

AP-125, Designing Microcontroller Systems for Electrically Noisy Environments	Order Number 210313
AP-445, 8XC196KR Peripherals: A User's Point of View	Order Number 270873
AP-449, A Comparison of the Event Processor Array (EPA) and High Speed Input/Output (HSIO) Unit	Order Number 270968

1.3.3 World Wide Web

We offer a variety of information on the World Wide Web. Use the following URLs to find information on our web site:

```
http://developer.intel.com/design/mcs96/
http://www.intel.com/
```

1.3.4 FaxBack* Service

FaxBack is an on-demand publishing system that sends documents to your fax machine. You can get product announcements, change notifications, product literature, device characteristics, design recommendations, and quality and reliability information from FaxBack 24 hours a day, 7 days a week.

1-800-628-2283	U.S. and Canada
916-356-3105	U.S., Canada, Japan, APac
44(0)1793-496646	Europe

Think of the FaxBack service as a library of technical documents that you can access with your phone. Just dial the telephone number and respond to the system prompts. After you select a document, the system sends a copy to your fax machine.

Each document is assigned an order number and is listed in a subject catalog. The first time you use FaxBack, you should order the appropriate subject catalogs to get a complete listing of document order numbers. Catalogs are updated twice monthly, so call for the latest information. The following catalogs and information are available at the time of publication:

1. *Solutions OEM* subscription form
2. Microcontroller and flash catalog
3. Development tools catalog
4. Systems catalog
5. Multimedia catalog
6. Multibus and iRMX® software catalog and BBS file listings
7. Microprocessor, PCI, and peripheral catalog
8. Quality and reliability and change notification catalog
9. iAL (Intel Architecture Labs) technology catalog

1.3.5 Bulletin Board System (BBS)

The bulletin board system (BBS) lets you download files to your computer. The application BBS has the latest *Ap*BUILDER software, hypertext manuals and datasheets, software drivers, firmware upgrades, application notes and utilities, and quality and reliability data.

916-356-3600	U.S., Canada, Japan, APac (up to 19.2 Kbaud)
44(0)1793-496340	Europe

The toll-free BBS (available in the U.S. and Canada) offers lists of documents available from FaxBack, a master list of files available from the application BBS, and a BBS user's guide. The BBS file listing is also available from FaxBack (see page 1-6 for phone numbers and a description of the FaxBack service).

1-800-897-2536 U.S. and Canada only

Any customer with a modem and computer can access the BBS. The system provides automatic configuration support for 1200- through 19200-baud modems. Typical modem settings are 14400 baud, no parity, 8 data bits, and 1 stop bit (14400, N, 8, 1).

To access the BBS, just dial the telephone number and respond to the system prompts. During your first session, the system asks you to register with the system operator by entering your name and location. The system operator will set up your access account within 24 hours. At that time, you can access the files on the BBS.

NOTE

If you encounter any difficulty accessing the high-speed modem, try the dedicated 2400-baud modem. Use these modem settings: 2400, N, 8, 1.

1.3.5.1 How to Find MCS 96 Microcontroller Files on the BBS

Application notes, utilities, and product literature are available from the BBS. To access the files, complete these steps:

1. Enter **F** from the BBS Main menu. The BBS displays the Intel Apps Files menu.
2. Type **L** and press <Enter>. The BBS displays the list of areas and prompts for the area number.
3. Type **12** and press <Enter> to select MCS 96 Family. The BBS displays a list of subject areas including general and product-specific subjects.
4. Type the number that corresponds to the subject of interest and press <Enter> to list the latest files.
5. Type the file numbers to select the files you wish to download (for example, **1,6** for files 1 and 6 or **3-7** for files 3, 4, 5, 6, and 7) and press <Enter>. The BBS displays the approximate time required to download the files you have selected and gives you the option to download them.

1.3.5.2 How to Find *Ap*BUILDER Software and Hypertext Documents on the BBS

The latest *Ap*BUILDER files and hypertext manuals and data sheets are available first from the BBS. To access the files, complete these steps:

1. Type **F** from the BBS Main menu. The BBS displays the Intel Apps Files menu.
2. Type **L** and press <Enter>. The BBS displays the list of areas and prompts for the area number.
3. Type **25** and press <Enter> to select *Ap*BUILDER/Hypertext. The BBS displays several options: one for *Ap*BUILDER software and the others for hypertext documents for specific product families.
4. Type **1** and press <Enter> to list the latest *Ap*BUILDER files or type **2** and press <Enter> to list the hypertext manuals and datasheets for MCS 96 microcontrollers.
5. Type the file numbers to select the files you wish to download (for example, **1,6** for files 1 and 6 or **3-7** for files 3, 4, 5, 6, and 7) and press <Enter>. The BBS displays the approximate time required to download the selected files and gives you the option to download them.

1.4 TECHNICAL SUPPORT

In the U.S. and Canada, technical support representatives are available to answer your questions between 5 a.m. and 5 p.m. PST. You can also fax your questions to us. (Please include your voice telephone number and indicate whether you prefer a response by phone or by fax). Outside the U.S. and Canada, please contact your local distributor.

1-800-628-8686	U.S. and Canada
916-356-7599	U.S. and Canada
916-356-6100 (fax)	U.S. and Canada

1.5 PRODUCT LITERATURE

You can order product literature from the following Intel literature centers.

1-800-468-8118	U.S. and Canada
708-296-9333	U.S. (from overseas)
44(0)1793-431155	Europe (U.K.)
44(0)1793-421333	Germany
44(0)1793-421777	France
81(0)120-47-88-32	Japan (fax only)

2

Getting Started with the 80C196Mx Demo Board

CHAPTER 2
GETTING STARTED WITH THE 80C196MX DEMO BOARD

The 80C196Mx demo board kit contains hardware and software to enable you to write, execute, monitor, and debug application software. This chapter includes a list of the 80C196Mx demo board kit contents. It previews the hardware and software design tools, and it steps you through the procedures for initializing and running the demo board. Following chapters describe the hardware and software in more detail.

2.1 DEMO BOARD KIT CONTENTS

The 80C196Mx Demo Board kit includes the following items:

- 80C196Mx Demo Board

  The 80C196Mx demo board ships with an N87C196MH embedded microcontroller in an 84-lead PLCC package installed. The N87C196MH features 32 KBytes of on-chip One Time Programmable Read Only Memory (OTPROM). The kit also ships with an N87C196MC and an N87C196MD microcontroller. The N87C196MC and N87C196MD support 16 Kbytes of on-chip OTPROM. The rismmx.exe monitor code (described below) is preprogrammed in all of the microcontroller modules' OTPROM.

  The board also includes 64 KBytes of SRAM for user code downloaded using ECM96Mx. The SRAM is resident in a standard 32-pin JEDEC socket and can be jumpered for two memory sizes ranging from 64K-256K of SRAM or 256K of EPROM.

- 3.5” MS-DOS* Diskette

  A 3.5” MS-DOS diskette contains software for running and debugging 80C196Mx programs from a host PC:

  — ecm96mx.exe
  rismmx.a96
  rismmx.lst
  rismmx.obj
  rismmx.hex

  ECM96MX and RISMMx comprise the embedded controller monitor software (ECM) for running and debugging your 80C196Mx programs. These programs are introduced in Chapter 4 and described in Chapters 5 and 6.

  — mxdemo.a96
  mxdemo.lst
  mxdemo.obj.

  MxDEMO is an 80C196Mx demonstration program. It is a convenient vehicle for experimenting with ECM96Mx and RISMMx commands.

  — cstart.a96
  cstart.obj

  CSTART maintains the chip-select configuration required by the ECM software. The object code file should be linked with user code that is loaded onto the 80C196Mx demo board.

  — hexobj.exe

  HEXOBJ converts HEX (hexadecimal) formatted diskette files to Intel object module file formats (OMFs), which can then be loaded by ECM96Mx (see “Loading and Saving Object Code” on page 5-5).

  — 80c196np.inc

  This file contains variable and macro definitions for rismmx.a96. The diskette file should be used when the RISMMx source code is modified and recompiled.

- Technical Documentation

  This manual provides you with the information needed to get up and running with the 80C196Mx demo board. For available related documentation, see “Related Documents and Products” on page 1-5. A set of demo board schematics is also provided in this manual.

2.2 APPLYING POWER TO THE DEMO BOARD

You must provide a +5 VDC power supply for the 80C196Mx demo board. It must be a regulated supply comparable to the ELPAC model WM113. (This and other models are available through DigiKey Corporation.)

Use the following procedure to power up the board:

1. Turn off power to the PC and the power supply.
2. Connect the serial port cable from the board's P1 connector to the com1 or com2 serial port on your PC.

   (The board-to-PC connection is not used until the Embedded Controller Monitor (ECM) is invoked. However, the connection should be made before power is supplied to the board. Figure A-1 on page A-2 shows the location of the demo board's power, ground, and serial port connections.)
3. Connect the power cable from the power supply to the JP2 connector on the 80C196Mx demo board.

WARNING

A regulated +5 VDC power supply must be used. Lower voltage might not operate the demo board. Higher voltage might damage the demo board. An unregulated power supply may cause unpredictable failure conditions.

Be sure that the plug from the power supply is oriented properly on the board. If it is plugged in backward, you may damage components on the board.

4. Turn on the PC and power supply. You should now observe the LED (light-emitting diode) bank at DP1 on the 80C196Mx demo board flashing through a power-up sequence. At power-on, LEDs 1 through 8 briefly turn on and off together. They then blink in sequence continuously under control of the RISMMx program in the EPROM. LEDs 9 and 10 remain off during the entire power-up sequence.

   If the LED bank is not flashing as described, check the following items:

   — Be sure that power is supplied to the board. Check the connection between the power supply cable and the board's power connector.

   — Confirm that the jumper settings are correct for the memory devices shipped with the board (or for a memory device that you have installed).

   — If you have changed a memory device on the board, check the speed of the device to ensure it meets specifications.

   — Press the reset button (S1) on the 80C196Mx demo board. If the board still does not respond, see "Technical Support" on page 1-8 for information on getting assistance.

2.3 INVOKING THE EMBEDDED CONTROLLER MONITOR SOFTWARE AND THE DEMONSTRATION PROGRAM

After the 80C196Mx demo board is initialized and executing RISMMx from the EPROM, you can start the Embedded Controller Monitor (ECM) and run the demonstration program.

1. Insert the distribution disk in the drive of your PC.
2. Create a directory for the embedded controller monitor software and copy the contents of the diskette to the directory. From this directory (for example, c:\ecm), type the following command at the DOS prompt:

   ecm96mx -baud 9600 <Enter>

   You can also execute directly from the diskette by entering the following command at the DOS prompt:

   d:\ecm96mx -baud 9600 <Enter>

   (If you don't use the d: drive, substitute the corresponding letter for your drive.)
3. Observe the ECM96Mx monitor screen displayed on your PC.

   When the ECM96Mx program is invoked, it communicates with the board and interrupts the RISMMx monitor. The continuous LED sequencing terminates, and a steady pattern is displayed. The ECM96Mx program displays the baud rate followed by an asterisk (*), which is the prompt for input. At this point you can use the ECM96Mx commands described in Chapter 5.
4. To download the demo board demonstration program from the PC to the SRAM on the demo board, type:

   load *path*\mxdemo.obj<Enter>
   go<Enter>

   where *path*\ represents the drive and directory where you installed the ECM96Mx software.

   You are now executing the 80C196Mx demonstration program within the ECM96Mx/RISMMx debugger environment. The LEDs now sequence in a new pattern.

   The MxDEMO program is a good vehicle for experimenting with the ECM96Mx commands (Chapter 5) and the RISMMx commands (Chapter 6). To return to DOS, type:

   exit <Enter>

3

Introduction to the 80C196Mx Demo Board

CHAPTER 3
INTRODUCTION TO THE 80C196MX DEMO BOARD

This chapter describes the 80C196Mx demo board. The board is designed as a basic demonstration system for evaluating hardware and software performance. This chapter also includes a block diagram of the board and a diagram of the major components of the board with a brief description of each functional section.

3.1 BLOCK AND COMPONENT DIAGRAMS OF THE BOARD

The 80C196Mx demo board is shipped with an 80C196MH device in the socket. The 80C196MH device can be replaced with an 80C196MC or an 80C196MD. For configuring the board for the preferred device refer to the jumper settings section of this user manual. The diagram illustrates the four main parts of the board: the 80C196Mx microcontroller, digital I/O, memory, and the interface between the 80C196Mx and the host PC. Following sections of this chapter describe these parts. The memory section can accommodate SRAM and EPROM (normal operation and programming). As shipped, the board has a 64-Kbyte SRAM.

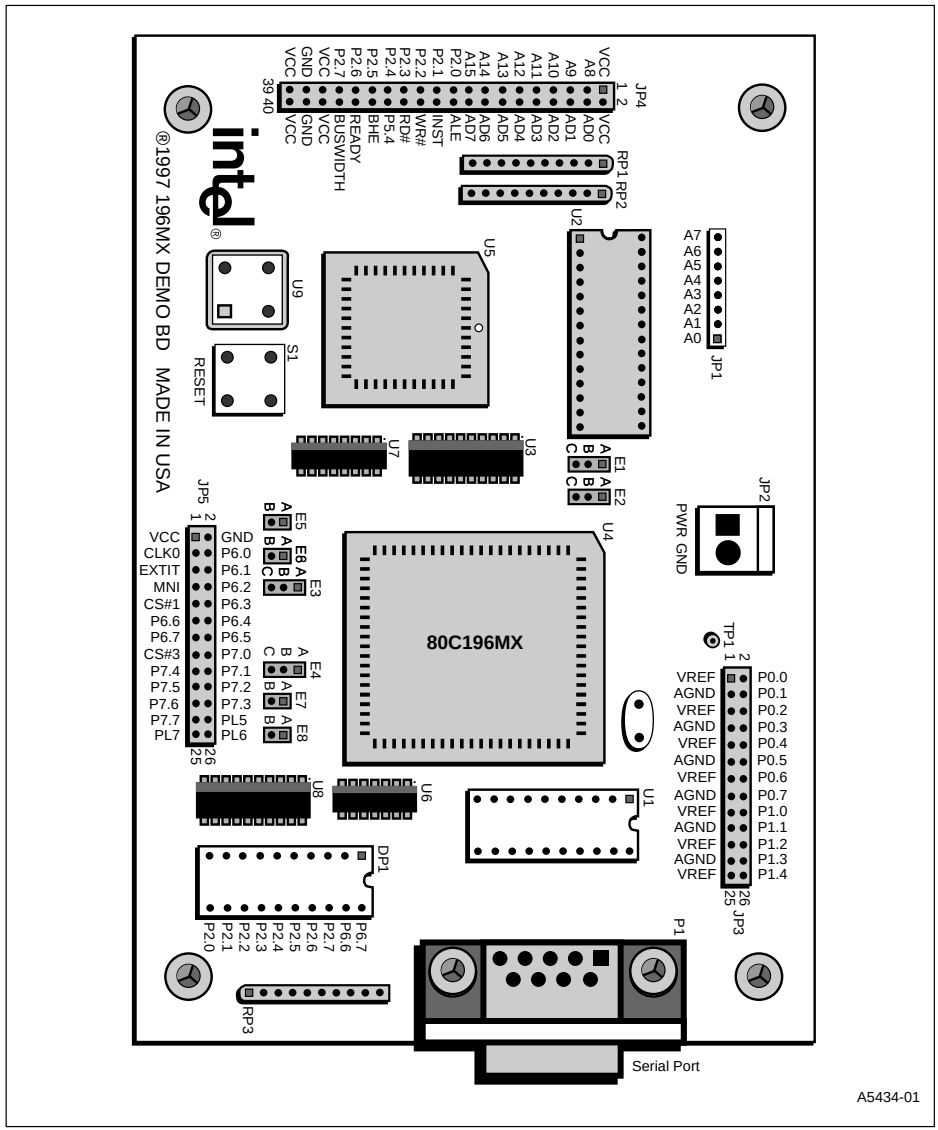


Figure 3-1. Component-level Diagram of the 80C196Mx Demo Board

Figure 3-1 is a component-level diagram of the demo board. Details of the components are given in Appendix A, “Components, Jumpers, and Connectors” and Appendix B, “Parts List.”

3.2 THE 80C196MC/MH/MD MICROCONTROLLER

The 8xC196Mx controllers are 16-bit microcontrollers, that are designed primarily to control three-phase AC induction and DC brushless motors. The 8xC196Mx microcontrollers are based on Intel's MCS® 96 architecture and are manufactured on Intel's CHMOS process.

The N87C196MH features an enhanced three-phase wave form generator specifically designed for use in inverter motor controller applications. This peripheral provides pulse-width modulation and three-phase sine wave generation with minimal CPU intervention. The N87C196MH also features two dedicated serial port peripherals, allowing reduced software overhead.

Refer to the *8XC196MC, 8XC196MD, 8XC196MH Microcontroller User's Manual* (order number 272181) for more information.

3.3 HOST INTERFACE

The host interface is a connection between the host PC serial port (com1 or com2) and the 80C196Mx serial I/O port (Figure 3-1). The com1 or com2 port connects to a 9-pin connector (P1) on the board and then to the 80C196Mx serial I/O port via an RS-232 interface (U1) and a UART (U5). The RS-232 interface uses the non-maskable interrupt (NMI) to signal the 80C196Mx that a character from the host is ready for reception.

3.4 80C196MX MEMORY SYSTEM

The 80C196Mx demo board is configured for 8-bit bus width. A key to using the 80C196Mx memory interface is understanding the relationship between internal memory addresses and external memory addresses. For details see the *8XC196MC, 8XC196MD, 8XC196MH Microcontroller User's Manual* (order number 272181).

3.5 MEMORY MODES

The Chip Configuration Registers (CCRs) support a variety of memory bus functions. The CCR bits select memory modes and are used for address/data bus control. Refer to the *8XC196MC, 8XC196MD, 8XC196MH Microcontroller User's Manual* (order number 272181) for information on CCR programming options.

3.6 USING SRAM AND EPROM

The 80C196Mx demo board supports SRAM (static operation) and EPROM (normal operations and programming). As shipped, the demo board has a 64-Kbyte SRAM in U2 (see Figure 3-1 on page 3-2), which can be replaced with a 256-Kbyte EPROM. At power-up, the 80C196Mx boots from the on-chip OTPROM. You can then download your application code to the SRAM. (See "Applying Power to the Demo Board" on page 2-2 and "Invoking the Embedded Controller Monitor Software and the Demonstration Program" on page 2-4.)

3.6.1 Memory Configurations and Installation

The demo board fully supports both SRAM and EPROM devices. Sockets U2 and U6 (Figure 3-1 on page 3-2) accept 28- and 32-pin DIP devices. The ECM96Mx/RISMMx software (Chapter 4, “Introduction to the Embedded Controller Monitor (ECM)”) limits the usable RAM size to 32 Kbytes.

The following procedure is for installing memory devices on the board.

1. Turn off power to the board.
2. Insert the memory device in socket U2.
3. Establish the jumper settings as shown in Table 3-1.

Table 3-1. Memory Configuration

E1	E2	Configuration For
B-C	A-B	64k and 256K RAM
A-B	B-C	256K EPROM

Note that the board ships with 64K (8K x 8 bits) or SRAM mapped at A000H to BFFFH. If 256K parts are installed, the decoding scheme used limits access to only 128K at 8000H to BFFFH.

4. If no external analog power reference is used, jumper E7 and E8.
5. If you wish to boot the device from external memory (not on the board), jumper E6.
6. If you are not using the on-board UART and wish to use its memory range (0000H-1FFFH) for an external device, remove the jumper on E5 and remove ICU5.
7. Power up the board according to the instructions in “Applying Power to the Demo Board” on page 2-2.
8. To load the ECM96Mx software, see “Invoking the Embedded Controller Monitor Software and the Demonstration Program” on page 2-4.

3.7 CHANGING THE MICROCONTROLLER MODULE

The demo board kit also includes an N87C196MC and an N87C196MD microcontroller. To change the microcontroller module on the board:

1. Use standard anti-static precautions such as wearing a ground strap.
2. Remove the N87C196MH microcontroller in socket U4 using an IC extractor.
3. Set the jumpers to configure the board for the type of processor that you are using.

Table 3-2. Processor Type Selection

E3	E4	Configuration For
A-B	A-B	196MC/MH
B-C	B-C	196MD

4. Power up the board according to the instructions in “Applying Power to the Demo Board” on page 2-2.

4

Introduction to the Embedded Controller Monitor (iECM)

CHAPTER 4
INTRODUCTION TO THE EMBEDDED CONTROLLER MONITOR (ECM)

This chapter introduces the Embedded Controller Monitor (ECM) user interface. This is the interface between the PC-resident software and the evaluation board firmware. The ECM software consists of two programs: ecm96mx.exe and mxr_main.hex. The commands for these programs are described in Chapter 5, “ECM96Mx Commands” and Chapter 6, “RISM Registers and Commands”.

4.1 EMBEDDED CONTROLLER MONITOR (ECM)

ECM is the software interface between the host system and the user code running on the evaluation board. It provides basic debug capabilities, including loading object files into system RAM, examining and modifying variables, and executing and stepping through code.

The 8xC196Mx evaluation board uses a version of the ECM written for the MCS® 96 microcontrollers with extended addressing capability. The ECM environment comprises two independent programs: mxr_main.hex and ecm96mx.exe. The mxr_main.hex program (referred to as RISMMX) resides in the evaluation board ROM; 80C196Mx executes it. The ecm96mx.exe software (known as ECM96Mx) resides and executes in DOS*- and Windows*-based PCs and BIOS-compatible computers.

RISMMx is a reduced instruction set monitor for the 80C196Mx. It executes rudimentary operations issued by ecm96mx.exe, which operates in the host PC. RISMMx consists of approximately 700 bytes of 80C196Mx code: a short section of initialization code and an interrupt service routine (ISR) that processes interrupts from the host system. The RISMMx ISR consists of a short prologue and then a case-jump to one of several handlers.

ECM96Mx, executing in the host PC, provides commands for loading and running code on the 80C196Mx. It also has features that facilitate test and debug tasks. For example, it can use include, list, and log files to record on-line ECM sessions and construct batch ECM sessions.

Partitioning the ECM into two separate programs supports a number of goals in developing this system:

- The RISMMx code in the evaluation board is simple and small. This maximizes the space available for user code.
- The ECM96Mx user interface's features expand beyond the resources of the evaluation board because ECM96Mx runs in the host PC.
- RISMMx and ECM96Mx run concurrently. They allow you to interrogate and modify the state of the evaluation board system while it is running.

4.2 RESTRICTIONS

The ECM operates under several restrictions:

- Several user stack words are reserved for RISMMx software use when the evaluation board processes a host interrupt (see the CAUTION on page 5-17). Internal register locations 0001E0H–000201H are reserved for RISMMx code use. Users must ensure that no registers in this partition are used by code operating with the RISMMx.
- A 9600-baud asynchronous serial port must be available on the host PC.
- The TRAP instruction is reserved.

5

ECM96MX Commands

CHAPTER 5
ECM96MX COMMANDS

This chapter describes the ECM96Mx commands. To begin using ECM96Mx, see the procedures for powering up the board (“Connecting the Evaluation Board to the Host System” on page 2-3) and invoking ECM96Mx for the first time (“Invoking the Embedded Controller Monitor Software” on page 2-4).

5.1 ECM DEFINED

ECM96Mx is the portion of the Embedded Controller Monitor (ECM) that runs on the host PC. It provides several tools with RISMMx for testing and debugging code on the evaluation board. ECM96Mx commands support tasks such as displaying and modifying program variables, initializing and operating program breakpoints, and single-stepping program execution.

5.2 COMMAND LINE NOTATION

This subsection explains command line notation. Even though the commands are listed in lowercase, both ECM96Mx and DOS are case-insensitive.

5.2.1 ECM96Mx Command Notation

When entering ECM96Mx commands, use the basic rules below (Table 5-1 includes examples of the rules):

- Use parameters and keywords when using commands that affect specific addresses and files.
- Use a comma as a Boolean OR. For example [this,that] is interpreted as [this] OR [that].
- Insert a hyphen immediately before the command when invoking ECM96Mx.

Table 5-1. ECM96Mx Command Notation

Rules	Example Command Line Notation and Descriptions†
Parameter	**Example**: string *byte_address* <Enter> **Parameter**: *byte_address* (used to specify a specific address)
Keyword	**Example**: go [from *code_address1* till *code_address2*] <Enter> **Keyword**: till (used to indicate a range. In this example, it indicates the range between the two parameters *codeaddress1* and *codeaddress2*.)

† The square brackets [ ] indicate an optional argument.

Table 5-1. ECM96Mx Command Notation (Continued)

Rules	Example Command Line Notation and Descriptions†
Comma	**Example**: dasm [*code_address*], [*count*] <Enter> **Comma**: used to separate distinct parameters. In this example, it separates the parameters [*code_address*] and [*count*].
Hyphenation	**Example**: ecm96mx [-com1(default), com2] <Enter> **Mandatory item**: A hyphen must precede the first command. **Description**: only used to invoke ECM96Mx.

† The square brackets [ ] indicate an optional argument.

5.2.2 DOS Command Rules

When entering DOS commands, follow these basic rules (Table 5-2 includes examples of the rules):

- Use parameters and keywords when using commands that affect specific addresses and files.
- Use commas to separate parameters.

Table 5-2. DOS Command Notation

Rules	Example Command Line Notation and Descriptions†
Parameter	**Example**: string *byte_address* <Enter> **Parameter**: *byte_address* (used to specify a specific address)
Keyword	**Example**: go [from *code_address1* till *code_address2*] <Enter> **Keyword**: till (used to indicate a range. In this example, it indicates the range between the two parameters *code_address1* and *code_address2*.)
Comma	**Example**: dasm [*code_address*], [*count*] <Enter> **Comma**: used to separate distinct parameters. In this example, it separates the parameters [*code_address*] and [*count*].

† The square brackets [ ] indicate an optional argument.

5.3 INITIALIZING AND TERMINATING ECM

The commands discussed in Table 5-3 invoke and terminate ECM96Mx from DOS, specify numerical bases (octal, decimal, or hexadecimal), and temporarily exit to DOS.

Table 5-3. Commands for Invoking and Terminating ECM96Mx

Command Names	Command Notations and Descriptions[1,2]
ecm96mx	**Notation**: ecm96mx [-*option1, option2, optionN*] <Enter> **Description**: Loads and executes the ECM96Mx software. Command options are described below. You can enter string options in any order; if the options are contradictory, the system accepts the last option entered. If ECM96Mx detects valid CTS (Clear to Send) and DSR (Data Set Ready) signals from the appropriate COM port, it signs on and displays one of the following command prompts: • When the board is executing code, it displays a greater-than sign (>). • When the board is **not** executing code, it displays an asterisk (*). • When CTS or DSR is not present, ECM96Mx notifies you and asks if you want to proceed or exit. If you proceed, ECM96Mx may operate properly, but your serial port or cabling may have a problem that will prevent proper operation.
com	**Notation**: [-com1 (default), com2] **Description**: Specifies the serial communication port to be used for host interface. The default is COM1.
baud	**Notation**:[-baud 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200] **Description**: Specifies the host-evaluation board communication rate. Baud rates higher than 9600 baud may not be supported on 8088-based PCs. A baud rate of 9600 baud can load 8 Kbytes of data in about 20 seconds. A baud rate of 57600 can load 8 Kbytes of data in about 4 seconds.
notypes	**Notation**: [-notypes] **Description**: Causes the object file loader to ignore type definition records in the object module. If it is invoked, the I/O routines recognize only basic data types, such as BYTEs, WORDs, and LONGs. More complex data types, such as PLM arrays and structures, are not recognized.

Notes:

1. All commands used to invoke ECM96Mx begin with a hyphen.
2. The square brackets [ ] indicate an optional argument.

5.4 GENERAL ECM96MX COMMANDS

Issue the general commands discussed in Table 5-4 after you invoke ECM96Mx.

Table 5-4. General ECM96Mx Commands

Command Names	Command Notations and Descriptions†
dos	**Notation**: dos <Enter> **Description**: Lets you temporarily leave ECM96Mx and return to DOS to run other application software. To return to ECM96Mx, type: exit <Enter>
exit	**Notation**: exit <Enter> **Description**: This command has two functions: • Returns the user to ECM96Mx from DOS. When it returns, ECM96Mx has the conditions that were in effect when it was temporarily suspended. • Closes any file that ECM96Mx has opened and exits to DOS. You can use this command even if the evaluation board is running a program (execution continues). ECM96Mx sets the selected COM port to 9600 baud, 8 data bits, no parity, and one stop bit, and then returns to DOS. The quit command also performs this duty.
base	**Notations**: • base <Enter> • base = *10t* • base [= 10o, 10t (breakpoint default), 10h (address default)] <Enter> **Description**: Displays the default arithmetic base. The default base is used to display variables and to enter numbers into the command parser. However, you can override the default base during input by adding an override character to the end of the number. The override characters are: o (octal), t (decimal), and h (hexadecimal). You must add the override character immediately after the last digit of the number. **Do not include a space**. Program addresses are always displayed in hexadecimal; and, breakpoint numbers are always displayed in decimal.
quit	**Notation**: quit <Enter> **Description**: Closes any file that ECM96Mx opened and exits to DOS. You can use this command even if the evaluation board is running a program (execution continues). ECM96Mx sets the selected COM port to 9600 baud, 8 data bits, no parity, and one stop bit, and then returns to DOS. The exit command also performs this duty.

† The square brackets [ ] indicate an optional argument.

5.5 FILE OPERATIONS

This section describes the commands that ECM96Mx uses to load and save object code, enter predefined strings of commands, log commands, and record entire debug sessions including user entries and the response generated by ECM96Mx on the host screen.

5.5.1 Loading and Saving Object Code

ECM96Mx accepts object files generated by Tasking (formerly BSO) development tools in the OMF96 version 3.0 format. ECM96Mx does not accept files containing unresolved externals or files containing relocateable records. Pass these files through the RL196 linker to resolve the externals and/or absolutely locate the relocateable segments.

To load new code from the PC into the 80C196Mx evaluation board, use the load and program operations. The load command downloads code that will reside in RAM.

Table 5-5 discusses the ECM96Mx commands that currently operate on object files.

Table 5-5. ECM96Mx Commands that Operate on Object Files

Command Names	Command Notations and Descriptions
load	**Notation**: load *filename* <Enter> **Description**: Loads the content records of the object *filename* into the evaluation board's code RAM or external RAM. The LOAD instruction cannot be used on ROM.
save	**Notations**: • save *filename* <Enter> • save *code_address1* to *code_address2* in *filename* <Enter> **Description**: Saves a region of memory as an object file that can be reloaded into the evaluation board's memory.

5.5.2 Include, Log, and List Files

Include files contain commands that ECM96Mx executes. You can prepare a command sequence off-line and later have ECM96Mx execute the commands just as if they were entered from the keyboard. An include file can be tedious to generate with a text editor. However, ECM96Mx can use a log file to store characters that you enter during an ECM96Mx session. Later, you can use the log file as an include file to recreate a command sequence. List files keep a running record of commands you enter and the response ECM96Mx generates.

You can insert comments in list and log files to make them easier to understand. A comment begins with a semicolon (;) and ends with an <Enter> or <Esc> character. The semicolon is part of the comment. The <Enter> or <Esc> character is not part of the comment.

When creating a log file, keep in mind you can place characters in the file to help you transform the file into a list file. You can use the list file to re-create command sequences. List files keep a running record of both the commands you enter and the responses ECM96Mx generates.

With the list file and log file commands, you can either overwrite existing data in the file or append data to the file. By using default filenames, you can gather list and log data in the default files and avoid having to create and manage a large number of separate files. ECM96Mx appends the date and time to log files and list files whenever they are opened. This information makes it easier for you to use a text editor to sort the data from the debug sessions.

The commands involved in include, log, and list operations are discussed in Table 5-6.

Table 5-6. Include, Log, and List Commands

Command Names	Command Notations and Descriptions
include	**Notation**: include *filename* <Enter> **Description**: Attempts to open *filename* as a read-only file. If the file can be opened, the command parser takes commands from that file. These commands must contain the exact sequence of ASCII characters you would type to execute them from the keyboard. Once the command parser reaches the end of the file, the file closes. Only one include file can be opened at a time.
pause	**Notation**: pause *filename* <Enter> **Description: (Use within include files**.) Pause is not a file-oriented command. When the command parser reads this command, it stops parsing and waits for you to press <Space> from the keyboard. The <Space> character cannot come from the include file. The pause command provides a way to pause in the middle of an include file operation. When you press <Space>, the parser continues parsing commands within the include file.
list	**Notations**: • list *filename* <Enter> • list <Enter> **Description**: Attempts to open *filename* as a writable file. If a file with *filename* already exists, ECM96Mx asks if the file is to be overwritten or if the new data should be appended to the end of the existing file. It then opens the file and stamps it with the current date and time from the system clock. After this, the file records the commands you enter and the responses ECM96Mx generates. If you do not enter a *filename*, the list command uses the last *filename* entered as part of a list *filename* command. If you have not entered any list *filename* commands, it uses the default filename “LIST.ECM”
listoff	**Notation**: listoff <Enter> **Description**: Closes the last list file specified by the list command. If no filename is specified, it uses the default filename “LIST.ECM”. ECM96Mx then stops recording new commands and responses.

Table 5-6. Include, Log, and List Commands (Continued)

Command Names	Command Notations and Descriptions
liston	**Notation**: liston <Enter> **Description**: Re-opens the list file in the append mode, so recording can start again. It also stamps the list file with the current date and time from the system clock. This stops new list information from being recorded.
log	**Notations**: • log *filename* <Enter> • log <Enter> **Description**: Attempts to open *filename* as a writable file. If a file with *filename* already exists, ECM96Mx asks if the file is to be overwritten or if the new data should be appended to the end of the file. It then opens the file and stamps it with the current data and time. After this, the file records the commands you enter. This file may contain nonprintable characters (e.g., <Esc>). If you do not enter a filename, the log command uses the last *filename* entered as part of a log *filename* command. If you have not entered any log *filename* commands, it uses the default filename "LOG.ECM"
logoff	**Notation**: logoff <Enter> **Description**: Closes a log file that has been specified by the log command. ECM96Mx then stops recording new commands.
logon	**Notation**: logon <Enter> **Description**: Re-opens the list file in the append mode, so recording can begin again. LOGON also stamps the list file with the current date and time from the system clock.

5.6 PROGRAM CONTROL

Commands in this group control program execution and allow you to reset the microcontroller, set execution breakpoints, start execution, stop execution, step, and super-step.

5.6.1 80C196Mx Reset

The following command resets the 80C196Mx without resetting the entire evaluation board.

reset **Notation**: reset chip <Enter>

Description: Physically resets the microcontroller by writing 0XXXX0001B to the RISM_DATA register. It then issues a "monitor_escape rism" command, which causes the evaluation board to execute a reset (RST) instruction.

5.6.2 Breakpoint Features

You can use breakpoints to stop execution at specified addresses. You may also use breakpoints to examine and/or modify registers and memory before resuming execution.

NOTE

When breakpoints are used to halt application code, microcontroller timers and peripherals (such as EPA, serial ports, and PWM) may remain active.

5.6.2.1 Breakpoint Operation

ECM96Mx provides 16 program execution breakpoints, BR0 to BR15, and a set of commands to set or clear the breakpoints. A command activates a breakpoint by assigning a specific address of an instruction where execution is to stop. For example, if "br2 = 0ff209dh <enter>", execution halts at address FF209DH. You must set the breakpoint to the address of the first (least significant) byte of the instruction. If a breakpoint is set to an address that is not the first byte of an instruction, execution is unpredictable.

To clear a breakpoint (make it "inactive") assign a zero to the breakpoint (e.g., "br2 = 0 <Enter>"). When execution begins, ECM96Mx saves the application code byte at any active breakpoint and substitutes a TRAP instruction for the saved byte. When the TRAP instruction executes, ECM96Mx restores the application code byte to its original address and decrements the application program counter to point at the restored instruction. The application code stops executing immediately **before** the instruction with a breakpoint. Two things happen on the screen when a break occurs:

- The prompt changes from a greater-than symbol (>) to an asterisk (*), indicating a halt condition has occurred.
- The target status (shown in the control panel at the top of the console screen) changes from "TARGET STATUS...RUNNING" to "TARGET STATUS...STOPPED".

Many monitor programs similar to ECM96Mx display a message on the console when a break occurs (e.g., "program break at 001234H"). However, ECM96Mx does not output a special break message. Because the system supports concurrent interrogation of the evaluation board on which the application code is running, a break can occur while you are displaying or modifying the state of the evaluation board. Special break messages interrupt command execution.

5.6.2.2 Breakpoint Commands

Breakpoint commands can display breakpoints while the application code is running or stopped. The commands can activate breakpoints only while the application code is stopped. Table 5-7 lists the breakpoint commands' notations and descriptions.

NOTE

When possible, avoid using BR0 and BR1 with the breakpoint command. The GO command with the TILL option can implicitly set BR0 and BR1 and thereby overwrite the addresses entered with the breakpoint command.

Table 5-7. Breakpoint Command Notations and Descriptions

Command Notations†	Command Descriptions
br <Enter>	Displays all active breakpoints (i.e., ≠ 0) or informs you that no breakpoints are active.
br [*bp_number* = *code_address*] <Enter>	Sets the breakpoint specified by *bp_number* to the value *code_address*. For example, to set breakpoint 3 to the address FF21A0H, type "br3=0ff21a0 <Enter>". (The BR command echoes this address as "21a0"; you can also enter the address FF21A0 as "21a0".) In this example, to clear the breakpoint, you would type "br3 = 0 <Enter>".
br [*bp_number*] <Enter>	Displays a breakpoint value and optionally changes the setting. ECM96Mx displays the setting of the selected breakpoint and waits for input. After typing (or not typing) a new value, you can press <Enter> or <Esc>: • <Enter> — Terminates the command. • <Esc> — Displays the next sequential breakpoint. Enter an address value to set the breakpoint or press <Esc> again to display the next breakpoint; the command wraps around from the last breakpoint (15) to the first breakpoint (0).

† The square brackets [ ] indicate an optional argument.

5.6.3 Program Execution Commands

The GO command and its options allow you to start and stop execution at specified addresses. You can execute this command only if the application code is stopped. In addition, a HALT command allows you to stop execution (when the application code is running).

The GO commands that set breakpoints use BP0 and BP1. Any break value already in one of these breakpoints is overwritten by the GO commands. As discussed in "Breakpoint Operation" on page 5-8, program execution stops just **before** execution of the instruction at the breakpoint address. ECM96Mx then temporarily deactivates that breakpoint when execution resumes (so the instruction can be executed) and finally reactivates the breakpoint. However, if execution stops at a breakpoint and no other breakpoint is set, the breakpoint is permanently deactivated, and you must use the HALT command to stop the application program.

Table 5-8 lists the GO and HALT commands' notations and descriptions.

Table 5-8. Go and Halt Command Notations and Descriptions

Command Names	Command Notations and Descriptions[1,2]
go	**Notation**: go <Enter> **Description**: Starts application code execution with the current value of the application's program counter (PC) and the current breakpoint array.
	Notation: go [forever] <Enter> **Description**: Clears the breakpoint array and starts execution at the current value of the application's PC.
	Notation: go [from *code_address*] <Enter> **Description**: Loads the application's PC with *code_address* and starts program code execution with the current breakpoint assignments.
	Notation: go [from *code_address* forever] <Enter> **Description**: Loads the application's PC with *code_address*, clears the breakpoint array, and begins program code execution.
	Notation: go [from *code_address1* till *code_address2*] <Enter> **Description**: Loads the application's PC with *code_address1*, sets the first default breakpoint (BP0) to the value of *code_address2*, and then begins program code execution.
	Notation: go [from *code_address1* till *code_address2* or *code_address3*] <Enter> **Description**: Functions like the previous command except that it also sets the second default breakpoint (BP1) to the value of *code_address3*.
	Notation: go [till *code_address*] <Enter> **Description**: Sets the first default breakpoint (BP0) to *code_address* and then begins the program code execution with the current setting of the application's PC and the breakpoint array.
	Notation: go [till *code_address1* or *code_address2*] <Enter> **Description**: Functions like the previous command except that it also sets the second default breakpoint (BP1) to the value of *code_address2*.

Notes:

1. Enter all hexadecimal addresses with a leading zero and no spaces (e.g., "0ff1209h").
2. The square brackets [ ] indicate an optional argument.

Table 5-8. Go and Halt Command Notations and Descriptions

Command Names	Command Notations and Descriptions[1,2]
halt	**Notation**: halt <Enter> **Description**: Stops program code execution by forcing the microcontroller to execute a jump-to-self instruction in a reserved location.

Notes:

1. Enter all hexadecimal addresses with a leading zero and no spaces (e.g., “0ff1209h”).
2. The square brackets [ ] indicate an optional argument.

5.6.4 Program Sequence Control

The ECM96Mx interface supports the instruction sequence commands necessary to single-step your application code. These commands are useful for testing and debugging short sections of code. This section defines the commands and certain limitations presented by this type of program flow control.

5.6.4.1 STEP/SUPER-STEP Operation

ECM96Mx provides STEP commands for executing code one instruction at a time. SUPER-STEP commands are similar, except they treat subroutines and interrupt service routines (ISRs) as single instructions. Between instructions, you can use ECM96Mx commands to check the states of the variables changed by the instruction to ensure that the program is operating properly. STEP commands allow a far more detailed view of program behavior. The disadvantage is that STEP commands do not occur in real time. This restriction makes it difficult or even impossible to use STEP commands with code that is dependent upon real-time events.

In some situations, STEP operations with enabled interrupt systems are confusing because interrupt service routines are also sequenced one instruction at a time. To avoid this problem, ECM96Mx artificially locks out interrupts with the basic STEP command operation.

SUPER-STEP is similar to STEP; however, SUPER-STEP interrupts are not artificially suppressed. An interrupt service routine or a subroutine call (and the body of the subroutine it calls) is treated as one indivisible instruction by the SUPER-STEP command. This allows you to ignore the details of subroutines and interrupt service routines while you view code operation. When an instruction uses SUPER-STEP, all service routines associated with enabled pending interrupts are executed. This allows limited stepping through code while operation continues in a concurrent environment; however, the system does not operate in real time. A better approach is to use the GO command to execute to a specified breakpoint and then STEP through the code.

ECM96Mx implements the STEP operation by using the TRAP instruction. To STEP over a given instruction, ECM96Mx determines the subsequent instruction (or all possible subsequent instructions for a conditional branch) and places a TRAP instruction at these locations. A TRAP is also set at location FF2080H in case the evaluation board is reset during the STEP. ECM96Mx allows the application program to execute until the program encounters TRAP locations. ECM96Mx then restores all overwritten application code bytes.

A SUPER-STEP operation is similar to a STEP; however, ECM96Mx treats the CALL instruction as a special case. During a STEP, ECM96Mx puts the TRAP at the evaluation board call address; during a SUPER-STEP, ECM96Mx places the TRAP at the instruction following the CALL. When the application's EI bit is saved, it suppresses interrupts during STEP (but not SUPER-STEP); then, ECM96Mx restores the interrupt. To ensure that the executed instruction does not modify the EI bit, ECM96Mx simulates several instructions (PUSHF, POPF, PUSHA, POPA, DI, EI) as opposed to the microcontroller executing the instructions. ECM96Mx also simulates the IDLPD instruction during a STEP to prevent the evaluation board from locking up. The simulation treats the IDLPD as a two-byte NOP. Instruction simulation occurs only with STEP operations. During a GO or a SUPER-STEP operation, the evaluation board executes all instructions.

5.6.4.2 STEP and SUPER-STEP Commands

ECM96Mx has four STEP commands and four corresponding SUPER-STEP commands. Aside from the interrupt operation differences discussed earlier, the STEP and SUPER-STEP commands behave the same way, so they are described here together. The command definition uses the phrase "single-step" instead of STEP or SUPER-STEP.

Table 5-9 lists the STEP and SUPER-STEP command notations and descriptions.

Table 5-9. STEP and SUPER-STEP Command Notation and Description

Command Notations†	Command Descriptions
[step | super-step] [-*option1, option2*] <Enter>	Single-steps your application code one instruction at a time.
[step | super-step] [*count*] <Enter>	Single-steps *count* times.
[step | super-step] [from *code_address*] <Enter>	Loads the application's PC with the value of *code_address* and then single-steps one time.
[step | super-step] [from *code_address, count*] <Enter>	Loads the application's PC with the value of *code_address* and then single-steps *count* times.

† The square brackets [ ] indicate an optional argument.

5.7 SUPPORTED DATA TYPES

ECM96Mx provides commands to display and modify program variables, including the following data types: BYTE, CHAR, WORD, DWORD, REAL, STACK, and STRING. ECM96Mx commands allow you to display variables or to initialize them either individually or as regions of memory that contain variables of the given type. ECM96Mx also supports microcontroller variables. You can examine the window select register (WSR); and you can examine and modify the program counter (PC), the program status word (PSW), and the stack pointer (SP).

NOTE

Memory locations 0001E0H–000201H are reserved for use by RISMMx. ECM96Mx gives a warning if you attempt to modify these memory locations.

Table 5-10 contains definitions for supported data types.

Table 5-10. Supported Data Types

Data Types	Data Type Definitions
byte	A BYTE is an 8-bit variable. No alignment rules are enforced for BYTE variables.
char	A CHAR is a special case of a BYTE. CHAR variables are displayed as ASCII characters.
word	A WORD is a 16-bit variable. The address of a WORD is the address of its least significant byte. A WORD must start at an even byte address.
dword	A DWORD is a 32-bit variable. The address of a DWORD is the address of its least significant byte. A DWORD must start on an address that is evenly divisible by four. This more restrictive alignment rule applies only to ECM96Mx commands when the single line assembler is used (see “Single Line Assembler (SLA) Commands” on page 5-17).
real	A REAL is a 32-bit binary floating-point number that conforms to the FPAL-96 definition. The 32 bits contain a sign bit, an 8-bit exponent field, and a 23-bit fraction field. ECM96Mx commands use standard scientific notation to represent REAL numbers. Note that FPAL-96 has special representations for +infinity and for NaNs (Not a Number, used to signal error conditions). If ECM96Mx detects one of these special values, it outputs an appropriate text string instead of trying to display the value in scientific notation.
stack	A STACK is a 16-bit variable that resides in the system stack. The address of a stack variable (*stack_address*) is relative to the current stack pointer and must be even word aligned.
string	A STRING is a sequence of ASCII characters terminated by the NUL character, which has the binary value of zero.

5.7.1 BYTE, WORD, DWORD, and REAL Commands

ECM96Mx has four basic commands to examine and modify BYTE, WORD, DWORD, and REAL variables. There is an additional command for WORD variables only.

Table 5-11 lists the BYTE, WORD, DWORD, and REAL commands' notations and descriptions.

Table 5-11. BYTE, WORD, DWORD, and REAL Command Notations

Command Notations[1,2]	Descriptions
variable [*variable_address*] <Enter>	Examine and possibly modify one or more variables at sequential addresses. ECM96Mx displays the hexadecimal address and the value of the variable in the default base. You can then terminate the command, modify the variable, or examine the variable at the next address: • <Enter> — Allows you to terminate the command. • *variable_value* — Assign this value to the variable. Allows you to terminate the command with <Enter> or examine the next variable by pressing <Esc>. • <Esc> — Allows you to examine the next variable. You can then terminate the command (<Enter>), assign a value (variable_value), or examine the next variable (<Esc>).
variable [*variable_address* = *variable_value*] <Enter>	Modify the value of a single variable.
variable [*variable_address* to *variable_address*] <Enter>	Examine the values of the variables in a range of addresses. In numerical form, ECM96Mx displays an address followed by up to 16 bytes of memory as BYTE, WORD, DWORD, or REAL values. To stop the output, press <Space>. To resume the output, press <Space> again. To terminate the command press <Enter>.
variable [*variable_address1* to *variable_address2* = *variable_value*] <Enter>	Initialize a region of memory to a given value. At 9600 baud, setting each value takes a little over one millisecond. To terminate the command press <Enter>; this leaves only a part of the memory region initialized.
word [*word_address1* to *word_address2* = *word_address3* to *word_address4*] <Enter>	Copy a block of memory from the second address range to the first address range. This command applies to WORD variables only. To terminate the command, press <Enter>; this leaves only a part of the memory region copied.

1. Replace the variable with BYTE, WORD, DWORD, or REAL (e.g., "word 0ff0080h = 0 <Enter>").
2. The square brackets [ ] indicate an optional argument.

5.7.2 STACK Commands

There are two commands for examining the stack. Both commands can be used whether the application program is running or stopped.

Table 5-12 lists the STACK command's notations and descriptions.

Table 5-12. Stack Command Notations and Descriptions

Command Notations†	Command Descriptions
stack *stack_address* <Enter>	Examine the 16-bit variable at a given offset from the stack pointer. ECM96Mx executes a "word *word_address*" command where *word_address* takes the value of the system stack pointer *stack_address*.
stack [*stack_address1* to *stack_address2*] <Enter>	Examine a sequence of 16-bit variables at a fixed offset in the system stack. ECM96Mx executes a "word *word_address1* to *word_address2*" command where both *word_address* fields are formed by adding the corresponding *stack_address* to the current value of the system stack pointer. Press <Space> to stop the output for a long display. Press <Space> again to resume output, or press <Enter> to terminate the command.

† The square brackets [ ] indicate an optional argument.

5.7.3 STRING Commands

There is only one form of the STRING command:

string — **Notation**: string *byte_address*

Description: The line begins with a hexadecimal display of *byte_address* followed by the NUL-terminated ASCII string starting at that address. For long strings, only the first 60 characters display. When trailing characters are stripped, decimal points (.) are substituted for the first three characters stripped.

5.7.4 Register Command Variables

You can read microcontroller variables at any time, but you can modify them only while the evaluation board program is stopped. With these commands you can display and load the program counter (PC), program status word (PSW), window select register (WSR), and stack pointer (SP). Display is in the default base.

Use the commands in Table 5-13 to access register variables associated with the microcontroller rather than with the program..

Table 5-13. Register Variable Notations and Descriptions

Register Names	Register Command Notations †
program counter	**Notations**: • pc <Enter> • pc [= *byte_address*] <Enter>
program status word	**Notations**: • psw <Enter> • psw [= *word_value*] <Enter>
window select register	**Notations**: • wsr <Enter> • wsr [= *word_value*] <Enter>
stack pointer	**Notations**: • sp <Enter> • sp [= *word_address*] <Enter>

† The square brackets [ ] indicate an optional argument.

5.7.5 Displaying and Modifying the Stack Pointer (SP)

RISMMx stores two words in the stack pointer area to retain the program counter (PC) and the program status word (PSW) during an ECM96Mx host interface interrupt. For this reason, when you display the stack pointer with the SP command or the STACK command, the displayed value is always offset by a value that compensates for the host interrupt overhead. This makes storing the host-interrupt related PC and PSW transparent at the evaluation board command level. However, you must allow for the extra stack space used when calculating total stack space requirements. This transparency is convenient but potentially confusing if you display the stack pointer with the SP command and then either view or directly modify location 18H (the internal register address of the stack pointer). It is recommended that you do not directly modify the stack pointer with internal register address 18H.

CAUTION

To avoid conflict with the evaluation board's stack operations, modify the stack pointer only with the SP command or by executing application code. Do not attempt to directly modify the stack pointer via register address 18H. (Specific implementations of the RISMMx may prevent you from overwriting register 18H and thereby force the use of the SP command.) Always use the SP or STACK command to manipulate the stack pointer.

5.8 ASSEMBLY AND DISASSEMBLY

ECM96Mx supports examining and modifying code memory using the standard mnemonics for the MCS® 96 assembler (ASM96). Although standard mnemonics are used, ECM96Mx does not build a symbol table of user symbols as assembly mnemonics are entered. This limits the software to operate as a single line assembler (SLA). References are never made to information entered on other lines. The SLA does not generate labels. The ECM96Mx SLA accepts mnemonics for all standard instructions that can be executed by the microcontroller. It does not accept "generic" instructions, such as BE or CALL, processed by ASM96 into standard instructions for MCS 96 microcontrollers. Neither does it accept JE, SCALL, or LCALL, which are the specific instructions understood by an MCS 96 microcontroller.

5.8.1 Single Line Assembler (SLA) Commands

The SLA is useful for assembling short code sequences to patch application code as it is tested. These on-line software routines are useful for testing or patching programs, but the tool is not intended as a replacement for a full-featured assembler (such as ASM96) working with an in-circuit emulator. You can invoke the SLA whether application code is being executed or not. It is not recommended that you dynamically modify code executed during your modification session.

Table 5-14 lists the SLA command's notations and descriptions.

Table 5-14. SLA Command Notations and Descriptions

Command Notations†	Command Descriptions
asm [*code_address*] <Enter>	Causes ECM96Mx to enter the SLA mode. The assembly program counter (APC) is set to *code_address*. Assembly language code, entered by the user, is converted to object code and loaded into the evaluation board's memory. ECM96Mx flags erroneous inputs but remains in the SLA mode. To terminate this mode, type "end <Enter>" (the only directive understood by the SLA).
asm <Enter>	Functions like the "asm *code_address* <Enter>" command except that the APC is not initialized. The first time the SLA is used, APC is set to FF2080H. Otherwise, APC points to the byte following the last instruction generated by the SLA.

† The square brackets [ ] indicate an optional argument.

5.8.2 Disassembly Commands

The disassembler converts binary object code in the evaluation board memory to ASM96 mnemonics. Use these commands for checking program patches or examining a portion of a program for which a listing is not available. You can use these commands whether application code is running or stopped.

Table 5-15 lists the disassembler command's notations and descriptions.

Table 5-15. Disassembler Command Notations and Descriptions

Command Notation†	Command Description
dasm <Enter>	Disassembles the instruction currently pointed to by the application's program counter (APC).
dasm [*count*] <Enter>	Reads the current value of the application's program counter (APC) and disassembles *count* instructions beginning at that location. The parameter *count* must be less than 256T (100H) so the command parser can distinguish this command from the command "dasm *code_address* <Enter>". (This restriction does not apply to the "dasm *code_address*, *count* <Enter>*"* instruction.) During lengthy displays, you can stop the output to the console by pressing <Space> and resume output by pressing <Space> again. Press <Enter> to terminate the command.
dasm [*code_address*] <Enter>	Disassembles the instruction at *code_address*. The parameter *code_address* must be greater than or equal to 256T (100H) so that the command parser can distinguish it from the "dasm *count* <Enter>*"* instruction.
dasm [-*code_address, count*] <Enter>	Disassembles *count* instructions starting with the instruction at *code_address*. During lengthy displays, you can stop the output to the console by pressing <Space> and resume output by pressing <Space> again. Press <Enter> to terminate the command.
dasm [*code_address* to *code_address*] <Enter>	Disassembles the region of memory specified. If an instruction crosses the ending address of the region, it is completely disassembled before the command terminates. During lengthy displays, you can stop the output to the console by pressing <Space> and resume output by pressing <Space> again. Press <Enter> to terminate the command.

† The square brackets [ ] indicate an optional argument.

6

RISMMx Commands

CHAPTER 6
RISM REGISTERS AND COMMANDS

This chapter describes the reduced instruction set monitor (RISM). The full RISM command set described in this chapter exists in the external ROM on the 80C196Mx target board. The target board runs this software under normal 80C196Mx operation.

6.1 RISM REGISTERS

Table 6-1 discusses RISM registers.

Table 6-1. RISM Registers

Registers	Definitions
RISM_DATA	A 32-bit register that acts as the primary data interface between software running in the host (PC) and the RISM running in the target (80C196Mx).
RISM_ADDR	A 24-bit register that contains the address to be used for reading and writing target memory.
RISM_STAT	An 8-bit register used to store RISM status and state information. This register contains the following Boolean flags: • DLE_FLAG: Indicates the next character received by the RISM should be treated as a data byte even if its value corresponds to an implemented command. • RUN_FLAG: Indicates that the target is running user code. • TRAP_FLAG: Indicates a software TRAP has occurred while running user code suspending its execution.
USER_PC	Saves the user's program counter while the user's code is not executing. Note that program execution must be stopped to use this command.
USER_PSW	Saves the user's program status word while the user's code is not executing.

6.2 RISM STRUCTURE

The RISM resides in the target system. It provides the interface between the target system and the user interface that resides in the host system. It is also compact and simple. This serves two purposes:

- The RISM can reside in a user's system with minimal impact on available memory.
- The RISM is easy to port into the target's environment.

The RISM internal state structure is simple: only three internal flags can change the way RISM deals with a character sent by the host.

- **DLE_FLAG:** When this flag is set, the next received character is assumed to be a data byte as opposed to a command byte.
- **RUN_FLAG:** This flag is set when the target is running user code. It can modify the operation of some RISM commands.
- **TRAP_FLAG:** This flag is set when the user code has been halted because the 80C196Mx executed a TRAP instruction. The TRAP_FLAG is cleared when the RISM starts the execution of user code.

6.3 RISM COMMAND DESCRIPTIONS

Table 6-2 on page 6-3 details the operation of each command sent to the RISM.

Table 6-2. RISM Command Descriptions

Value	Command	Description
00H	SET_DLE_FLAG	Sets the DLE flag in bit 0 of the MODE register to tell the RISM the next byte on the serial port is data that should be loaded into the DATA register. The flag is cleared as soon as the byte is read.
02H	TRANSMIT	Transmits the low byte of the DATA register to the serial port through the CHAR register, shifts the DATA register right (long) by eight bits, and increments ADDR by one. **ADDR** / **DATA** / **SBUF_TX** Before command: ADDR FF 22 14; DATA 7A 2F 80 67; SBUF_TX After command: ADDR FF 22 15; DATA 00 7A 2F 80; SBUF_TX 67
04H	READ_BYTE	Puts the contents of the (byte) memory address pointed to by the ADDR register into the low byte of the DATA register. **ADDR** / **DATA** / **Memory Addr. 2215 2214** Before command: ADDR FF 22 14; DATA; 2215 80, 2214 67 After command: ADDR FF 22 14; DATA 67; 2215 80, 2214 67
05H	READ_WORD	Puts the contents of the (word) memory address pointed to by the ADDR register into the low byte of the DATA register. **ADDR** / **DATA** / **Memory Addr. 2215 2214** Before command: ADDR 22 14; DATA; 2215 80, 2214 67 After command: ADDR 22 14; DATA 80 67; 2215 80, 2214 67
06H	READ_DOUBLE	Reads the double-word of memory pointed to the address register and places the results in the RISM_DATA register.

<table>
<tr><th>Value</th><th>Command</th><th>Description</th></tr>
<tr><td>07H</td><td>WRITE_BYTE</td><td>Puts the low byte of the DATA register into the memory address pointed to by the ADDR register and increments ADDR by one.

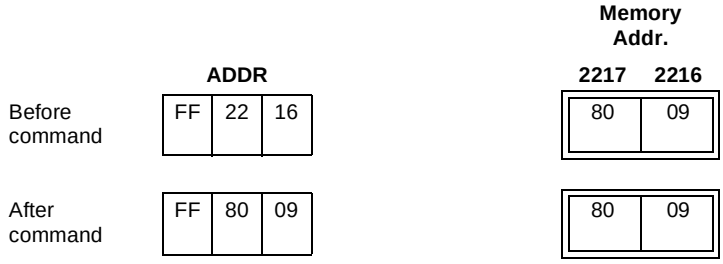
<table>
<tr><th></th><th colspan="3">ADDR</th><th colspan="4">DATA</th><th colspan="2">Memory Addr.
2217 2216</th></tr>
<tr><td>Before command</td><td>FF</td><td>22</td><td>16</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>FF</td><td>FF</td></tr>
<tr><td>After command</td><td>FF</td><td>22</td><td>17</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>FF</td><td>09</td></tr>
</table>

NOTE: To write to an OTPROM location, V_{PP} must be at +12.5 VDC. To write to an internal RAM location, V_{PP} can be at either +5.0 or +12.5 VDC.</td></tr>
<tr><td>08H</td><td>WRITE_WORD</td><td>Puts the low word of the DATA register into the memory address pointed to by the ADDR register and increments ADDR by two.

<table>
<tr><th></th><th colspan="3">ADDR</th><th colspan="4">DATA</th><th colspan="2">Memory Addr.
2217 2216</th></tr>
<tr><td>Before command</td><td>FF</td><td>22</td><td>16</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>FF</td><td>FF</td></tr>
<tr><td>After command</td><td>FF</td><td>22</td><td>18</td><td>2E</td><td>11</td><td>80</td><td>09</td><td>80</td><td>09</td></tr>
</table>

NOTE
To write to an OTPROM location, V_{PP} must be at +12.5 VDC. To write to an internal RAM location, V_{PP} can be at either +5.0 or +12.5 VDC.</td></tr>
<tr><td>09H</td><td>WRITE_DOUBLE</td><td>Stores the RISM_DATA register in the double-word of memory pointed to by the RISM_ADDR register and increments the RISM_ADDR register (by four) to point at the next memory double-word.</td></tr>
<tr><td>0AH</td><td>LOAD_ADDRESS</td><td>Puts the low word of the DATA register into the ADDR register.

<table>
<tr><th></th><th colspan="3">ADDR</th><th colspan="4">DATA</th></tr>
<tr><td>Before command</td><td>FF</td><td></td><td></td><td>F1</td><td>05</td><td>22</td><td>16</td></tr>
<tr><td>After command</td><td>FF</td><td>22</td><td>16</td><td>F1</td><td>05</td><td>22</td><td>16</td></tr>
</table>
</td></tr>
</table>

Value	Command	Description
0BH	INDIRECT _ADDRESS	Puts the word from the memory address pointed to by the ADDR register into the ADDR register. 
0CH	READ_PSW	Loads the RISM_DATA register with the PSW (Program Status Word) associated with the user's code. Most RISM implementations must check RUN_FLAG to determine how to access the user's PSW.
0DH	WRITE_PSW	Loads the PSW (Program Status Word) associated with the user's code from the RISM_DATA register. The host software will only invoke this command while user code is not running.
0EH	READ_SP	Loads the RISM_DATA register with the SP (Stack Pointer) associated with the user's code.
0FH	WRITE_SP	Loads the SP (Stack Pointer) from the RISM_DATA register. This command also pushes two values into the newly created stack area. These values are the PC (first) and PSW (second) associated with the idle loop which executes while user code is not running. The host software will only invoke this command while user code is not running.
10H	READ_PC	Loads the RISM_DATA register with the PC (Program Counter) associated with the user's code. Most RISM implementations will have to check RUN_FLAG to determine how to access the user's PC.
11H	WRITE_PC	Loads the PC (Program Counter) associated with the user's code from the RISM_DATA register. The host software will only invoke this command while user code is not running.
12H	START_USER	PUSHes the user PC, PSW, and WSR onto the stack and starts the application program from the location contained in the user PC. The RISM PC, PSW, and WSR will also be in the stack, so allow enough room on the stack for all six words. You can interrogate memory locations while your program is running. The RISM interrupts your program to process the command and then returns execution to your program.
13H	STOP_USER	Halts execution of the application program, POPs the user PC, PSW, and WSR from the stack, and PUSHes the RISM PC, PSW, and WSR back onto the stack. The RISM PC contains the location of the Monitor_Pause routine, so the RISM returns to Monitor_Pause.
	TRAP_ISR	A pseudo command that cannot be issued directly by the host software. It is executed when a TRAP instruction is executed. The TRAP instruction is used by ECM to implement software breakpoints and single stepping. On the 80C196Mx target board, these functions are supported for code execution from on-chip code RAM or the external RAM (cannot insert TRAP into ROM).

Value	Command	Description
14H	REPORT _STATUS	Loads a value into the DATA register. This value indicates the status of the application program: **Value** **Status** 00 halted 01 running 02 trapped
15H	MONITOR _ESCAPE	Provides for the addition of RISM commands for special purposes; it uses the RISM_DATA register to extend the command set of the RISM. If the value of the lower 16 bits of the RISM_DATA register is one (RISM_DATA = 0XXXX0001H) then the evaluation board microcontroller should execute either a reset (RST) instruction or a software initialization routine. The basic RISM requires only one of these “extended” commands.
16H	READ_WSR	Reads the value in the Windows Selection Register and puts it into the RISM_DATA register.
17H	WRITE_WSR	Takes the value in the RISM_DATA register and puts the data into the WSR. It also resets RI and INT_MASK1A.
18H	SET_BAUD	Takes the value in the RISM_DATA register and compares it to 7. If the value is less than or equal to 7, then it is assumed to be an index into the BAUD_RATE_CAPAB table. Otherwise the value is assumed to be the literal baud rate value.
19H	READ_WORD _TRANSMIT	Copies the address stored in RISM_ADDR to RISM_DATA when transmits that back to the host.
1AH	SET_BLK_FLAG	Is used in conjunction with block copy. When this flag is set, it assumes that the next N words are data and will be stored at the pre-determined starting point.
1BH	READ _CHECKSUM	Reads the checksum register and puts the data in RISM_DATA. After the command is executed, the checksum register is cleared/reset.

A

Components, Jumpers, and Connectors

APPENDIX A
COMPONENTS, JUMPERS, AND CONNECTORS

This appendix includes figures and tables to increase your understanding of the 80C196Mx demo board.

A.1 COMPONENTS

Table A-1 lists the major components found on the 80C196Mx demo board. See Figure A-1 for the location of each component. Refer to Appendix B-1for a complete list of parts used to build the board.

Table A-1. Component List

Component Label	Component Name
JP1	8-pin I/O Expansion
JP2	Power Supply Connector
JP3	26-pin I/O Expansion
JP4	40-pin I/O Expansion
JP5	26-pin I/O Expansion
U1	MAX233
U2	28/32-pin JEDEC SRAM
U3	74HC373
U4	84-pin PLCC N87C196MH
U5	16C550
U6	74HC14
U7	74HC14
U8	74HC240
U9	1.8432 MHz Canned Oscillator
Y1	16 MHz Canned Oscillator
P1	Host Serial Port
S1	Board Reset Switch
DP1	LED Bank
RP1	SIP Resistor Pack
RP2	SIP Resistor Pack
RP3	SIP Resistor Pack

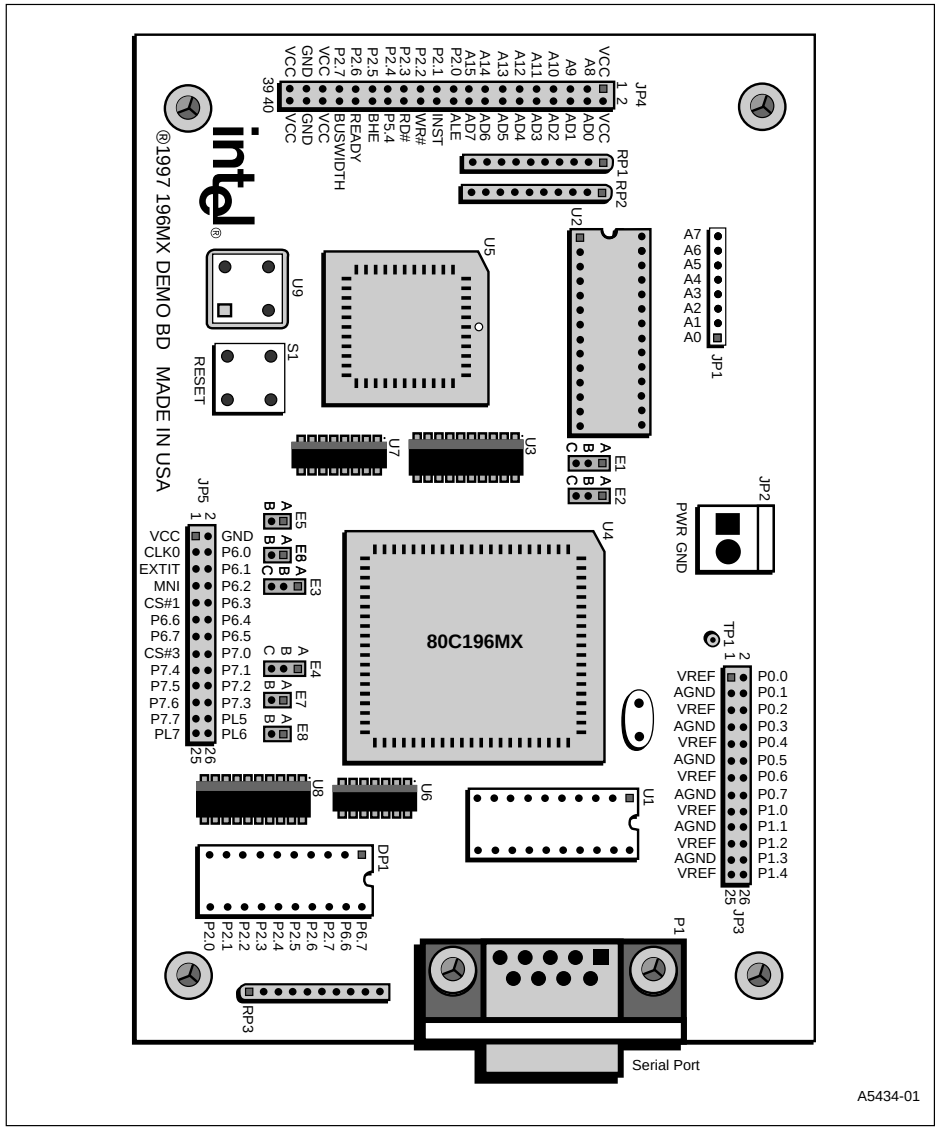


Figure A-1. 80C196Mx Demo Board Diagram

A.2 JUMPER DEFINITIONS

Table A-2 lists the definitions for jumpers on the 80C196Mx Demo Board.

Table A-2. Jumper Definitions

Jumper Label	Pin Number	Jumper Options
E1	U2:Pin 1	A-B = VCC B-C = A14
E2	U7:Pin2	A-B = P5.2 B-C = A14
E3	U4:P1.6	A-B = GND B-C = P1.6
E4	U4:P1.5	A-B = GND B-C = P1.5
E5	UARTINT	A-B = UARTINT
E6	GND	A-B = GND
E7	VREF	A-B = VCC
E8	ANGND	A-B = AGND

A.2.1 Memory Configuration Jumpers

Table A-3. Memory Configuration

E1	E2	Configuration For
B-C	A-B	64k and 256K RAM
A-B	B-C	256K EPROM

Note that the board ships with 64K (8K x 8 bits) or SRAM mapped at A000H to BFFFH. If 256K parts are installed, the decoding scheme used limits access to only 128K at 8000H to BFFFH.

A.2.2 Analog Power Reference Configuration

If no external analog power reference is used, jumper E7 and E8.

A.2.3 External Address Capability

If you wish to boot the device from external memory (not on the board), remove U5.

A.2.4 Chip-Dependent Jumpers

Table A-4. Processor Type Selection

E3	E4	Configuration For
A-B	A-B	196MC/MH
B-C	B-C	196MD

A.2.5 UART Interrupt

If you are not using the on-board UART and wish to use its memory range (0000H-1FFFH) for an external device, remove the jumper on E5.

A.3 POWER SUPPLY CONNECTOR JP2

The flag on the JP2 connector is oriented to the edge of the board. Figure A-2 depicts the orientation of the terminals with respect to the demo board.

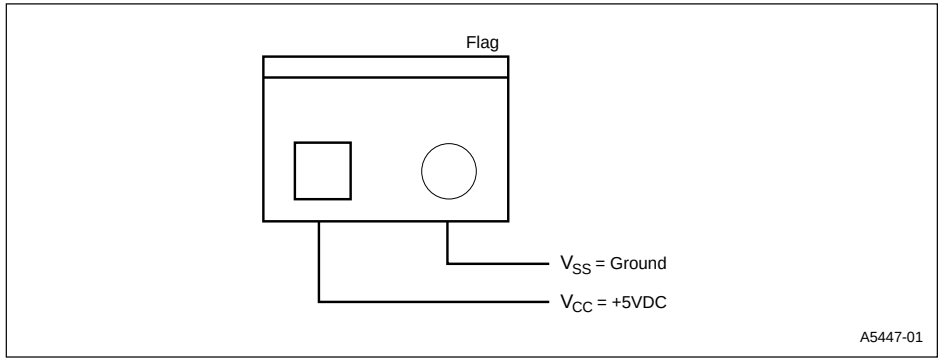


Figure A-2. Power Supply Connector JP2

A.4 I/O EXPANSION CONNECTORS JP1, JP3-5

The tables that follow describe the functions of I/O expansion connectors JP1 and JP3-5.

Table A-5. 8-pin I/O Expansion Connector JP1

Pin No	Function Name	Type	Description
1	A0	I/O	Address Lines 0:7. Address lines for de-multiplexed address bus
2	A1	I/O	
3	A2	I/O	
4	A3	I/O	
5	A4	I/O	
6	A5	I/O	
7	A6	I/O	
8	A7	I/O	

Table A-6. 26-pin I/O Expansion Connector JP3 (Sheet 1 of 2)

Pin No	Function Name	Type	Description
1	VREF	PWR	Reference Voltage for the A/D Converter
2	P0.0	I	Analog inputs to the on-chip A/D converter
3	AGND	GND	Reference Ground for the A/D Converter and Port 0 logic
4	P0.1	I	Analog inputs to the on-chip A/D converter
5	VREF	PWR	Reference Voltage for the A/D Converter
6	P0.2	I	Analog inputs to the on-chip A/D converter
7	AGND	GND	Reference Ground for the A/D Converter and Port 0 logic
8	P0.3	I	Analog inputs to the on-chip A/D converter
9	VREF	PWR	Reference Voltage for the A/D Converter
10	P0.4	I	Analog inputs to the on-chip A/D converter
11	AGND	GND	Reference Ground for the A/D Converter and Port 0 logic
12	P0.5	I	Analog inputs to the on-chip A/D converter
13	VREF	PWR	Reference Voltage for the A/D Converter
14	P0.6	I	Analog inputs to the on-chip A/D converter
15	AGND	GND	Reference Ground for the A/D Converter and Port 0 logic
16	P0.7	I	Analog inputs to the on-chip A/D converter
17	VREF	PWR	Reference Voltage for the A/D Converter
18	P1.0	I	Analog inputs to the on-chip A/D converter
19	AGND	GND	Reference Ground for the A/D Converter and Port 0 logic
20	P1.1	I	Analog inputs to the on-chip A/D converter
21	VREF	PWR	Reference Voltage for the A/D Converter
22	P1.2	I	Analog inputs to the on-chip A/D converter

Table A-6. 26-pin I/O Expansion Connector JP3 (Sheet 2 of 2)

23	AGND	GND	Reference Ground for the A/D Converter and Port 0 logic
24	P1.3	I	Analog inputs to the on-chip A/D converter
25	VREF	PWR	Reference Voltage for the A/D Converter
26	P1.4	I	Analog inputs to the on-chip A/D converter

Table A-7. 40-pin I/O Expansion Connector JP4 (Sheet 1 of 2)

Pin No	Function Name	Type	Description
1	VCC	PWR	Digital Supply Voltage (+5 VDC)
2	VCC	PWR	Digital Supply Voltage (+5 VDC)
3	A8	I/O	Address/Data lines for multiplexed address and data bus
4	AD0	I/O	Address/Data lines for multiplexed address and data bus
5	A9	I/O	Address/Data lines for multiplexed address and data bus
6	AD1	I/O	Address/Data lines for multiplexed address and data bus
7	A10	I/O	Address/Data lines for multiplexed address and data bus
8	AD2	I/O	Address/Data lines for multiplexed address and data bus
9	A11	I/O	Address/Data lines for multiplexed address and data bus
10	AD3	I/O	Address/Data lines for multiplexed address and data bus
11	A12	I/O	Address/Data lines for multiplexed address and data bus
12	AD4	I/O	Address/Data lines for multiplexed address and data bus
13	A13	I/O	Address/Data lines for multiplexed address and data bus
14	AD5	I/O	Address/Data lines for multiplexed address and data bus
15	A14	I/O	Address/Data lines for multiplexed address and data bus
16	AD6	I/O	Address/Data lines for multiplexed address and data bus
17	A15	I/O	Address/Data lines for multiplexed address and data bus
18	AD7	I/O	Address/Data lines for multiplexed address and data bus
19	P2.0	I/O	Bi-directional standard I/O port
20	ALE	O	Address Latch Enable output pin
21	P2.1	I/O	Bi-directional standard I/O port
22	INST	O	Instruction Fetch indicates instruction being fetched from external memory
23	P2.2	I/O	Bi-directional standard I/O port
24	WR#	O	Write indicates external write occurring when active low
25	P2.3	I/O	Bi-directional standard I/O port
26	RD#	O	Read asserted during external memory read
27	P2.4	I/O	Bi-directional standard I/O port
28	P5.4	I/O	Multiplexed with ONCE pin (configure as output)
29	P2.5	I/O	Bi-directional standard I/O port
30	BHE	O	Byte High Enable indicates that valid data is being transferred over the upper half of the system address/data bus

Table A-7. 40-pin I/O Expansion Connector JP4 (Sheet 2 of 2)

31	P2.6	I/O	Bi-directional standard I/O port
32	READY	I	Ready is used to lengthen external memory cycles for slow memory by generating wait states
33	P2.7	I/O	Bi-directional standard I/O port
34	BUSWIDTH	I	Configured in CCB for 16-bit bus cycle or 8-bit bus cycle
35	VCC	PWR	Digital Supply Voltage (+5 VDC)
36	VCC	PWR	Digital Supply Voltage (+5 VDC)
37	GND	GND	Digital Circuit Ground (0 V)
38	GND	GND	Digital Circuit Ground (0 V)
39	VCC	PWR	Digital Supply Voltage (+5 VDC)
40	VCC	PWR	Digital Supply Voltage (+5 VDC)

Table A-8. 26-pin I/O Expansion Connector JP5 (Sheet 1 of 2)

Pin No	Function Name	Type	Description
1	VCC	PWR	Digital Supply Voltage (+5 VDC)
2	GND	GND	Digital Circuit Ground (0 V)
3	CLKO	O	Output of Internal Clock Generator
4	P6.0	O	Wave Generator output or standard output port
5	EXTIT	I	Programmable Interrupt pin
6	P6.1	O	Wave Generator output or standard output port
7	NMI	I	Non-Maskable Interrupt pin
8	P6.2	O	Wave Generator output or standard output port
9	CS#1	I	Chip Select Address range 4000H-7FFFH
10	P6.3	O	Wave Generator output or standard output port
11	P6.6	O	Pulse Width Modulator output port
12	P6.4	O	Wave Generator output or standard output port
13	P6.7	O	Pulse Width Modulator output port
14	P6.5	O	Wave Generator output or standard output port
15	CS#3	I	Chip Select Address range C000H-FFFFH
16	P7.0	I/O	EPA Capture/Compare or standard I/O pins
17	P7.4	I/O	EPA Capture/Compare or standard I/O pins
18	P7.1	I/O	EPA Capture/Compare or standard I/O pins
19	P7.5	I/O	EPA Capture/Compare or standard I/O pins
20	P7.2	I/O	EPA Capture/Compare or standard I/O pins
21	P7.6	I/O	EPA Capture/Compare or standard I/O pins
22	P7.3	I/O	EPA Capture/Compare or standard I/O pins
23	P7.7	I/O	EPA Capture/Compare or standard I/O pins

Table A-8. 26-pin I/O Expansion Connector JP5 (Sheet 2 of 2)

24	P1.5	I	Analog inputs to the on-chip A/D converter or standard input port
25	P1.7	I	Analog inputs to the on-chip A/D converter or standard input port
26	P1.6	I	Analog inputs to the on-chip A/D converter or standard input port

A.5 LED BANK DESCRIPTIONS

At power-on and whenever the board is reset, LEDs 1 through 8 turn on then off together (see Figure A-3). Then they blink on in sequence continuously until the host PC sends a command to the board, power is turned off, or the board is reset. LED 9 remains off during the entire power-up sequence. LEDs 1 through 8 can be programmed to display port 1.7:0.

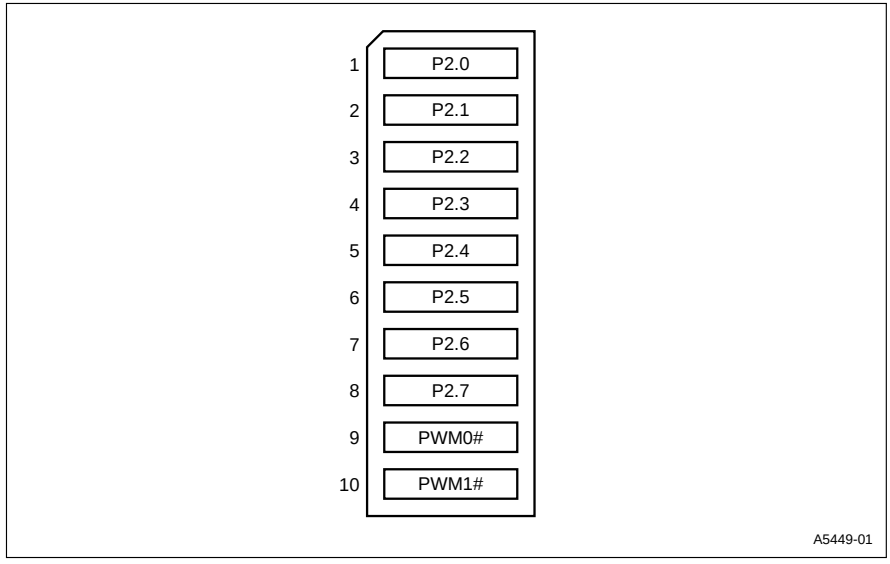


Figure A-3. LED Bank DP1

A.6 25-PIN TO 9-PIN RS-232 INTERFACE

If your computer has a 25-pin serial port connector, we recommend you buy a standard RS-232 25-pin to 9-pin conversion adapter or cable. Figure A-4 on page A-9 shows you how to assemble a 25-pin to 9-pin serial port interface adapter cable for the correct connection to the 80C196Mx demo board.

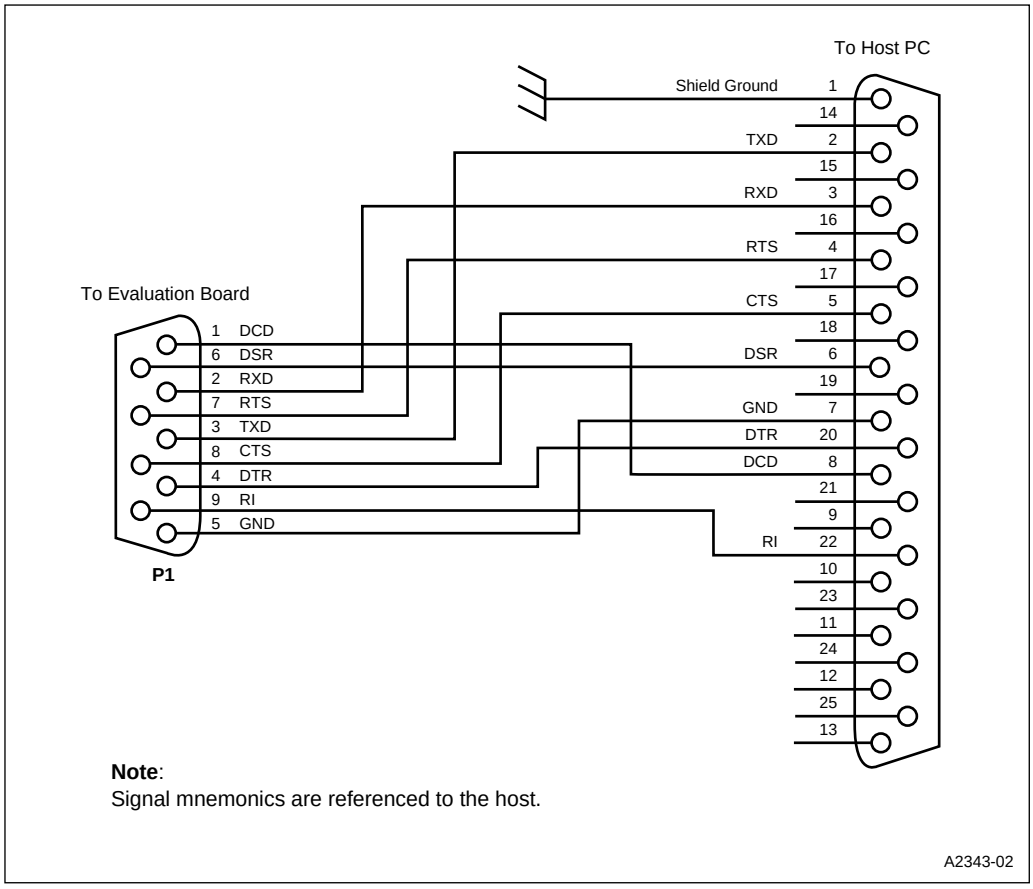


Figure A-4. Serial Interface

A.7 EXTERNAL MEMORY MAP

Table A-9 shows the area in the external memory map that by default are allocated.

Table A-9. External Memory Map

External Address Range	Allocation
C000H - FFFFH	Available Chips Select
A000H - BFFFH	SRAM Standard
8000H - 9FFFFH	SRAM Expansion
2000H - 7FFFH	Unallocated
0000H - 1FFFH	UART at 1E00H

B

Parts List

APPENDIX B
PARTS LIST

This appendix provides a list of all discrete and active components for the 80C196Mx demo board.

Table B-1. Parts List

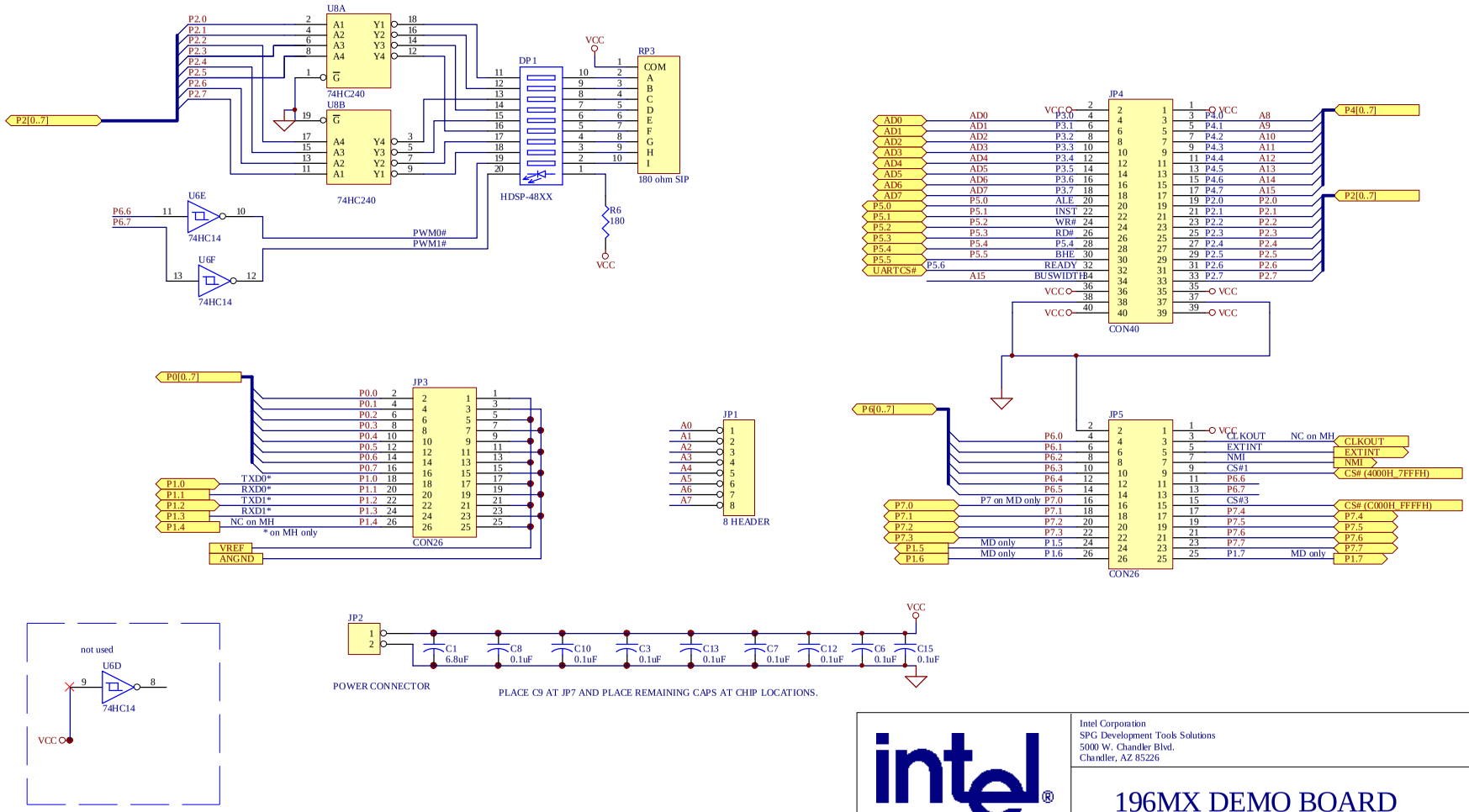
Item #	Qty.	Description		Designators
1	9	0.1µF Cap		C3, C6, C7, C8, C9, C10, C12, C13, C15
2	2	1.0µF Cap		C2, C11
3	1	1.8432 MHz Crystal		U9
4	2	1N4148 Diode		D1, D2
5	4	3-Way Jumper		E1, E2, E3, E4 (3-pin)
6	1	6.8 µF		C1
7	1	8 Header		JP1
8	1	8xC196Mx IC, Microcontroller		U4
9	5	10K Res		R1, R2, R3, R4, R5
10	1	10µF Cap		C14
11	1	16.0 MHz Crystal		Y1
12	1	16C550, IC UART		U5
13	2	30K SIP, Res Pack		RP1, RP2
14	2	30pF Cap		C4, C5
15	1	74HC14 IC, Logic		U6
16	1	74HC139 IC, Logic		U7
17	1	74HC240 IC, Logic		U8
18	1	74HC373 IC, Logic		U3
19	2	100K Res		R7, R8
20	1	180 Res		R6
21	1	180 ohm SIP, Res Pack		RP3
22	1	7164 IC, SRAM		U2
23	2	CON26		JP3, JP5
24	1	CON40		JP4
25	1	DP9 Female		P1

Table B-1. Parts List (Continued)

Item #	Qty.	Description		Designators
26	1	HDSP-48XX LED Display		DP1
27	1	MAX233 IC, RS232		U1
28	1	Power Connector		JP2
29	1	Reset Switch		S1
30	4	STD JMPR		E5, E6, E7, E8 (2-pin)
31	1	TP		TP1

C

Schematics



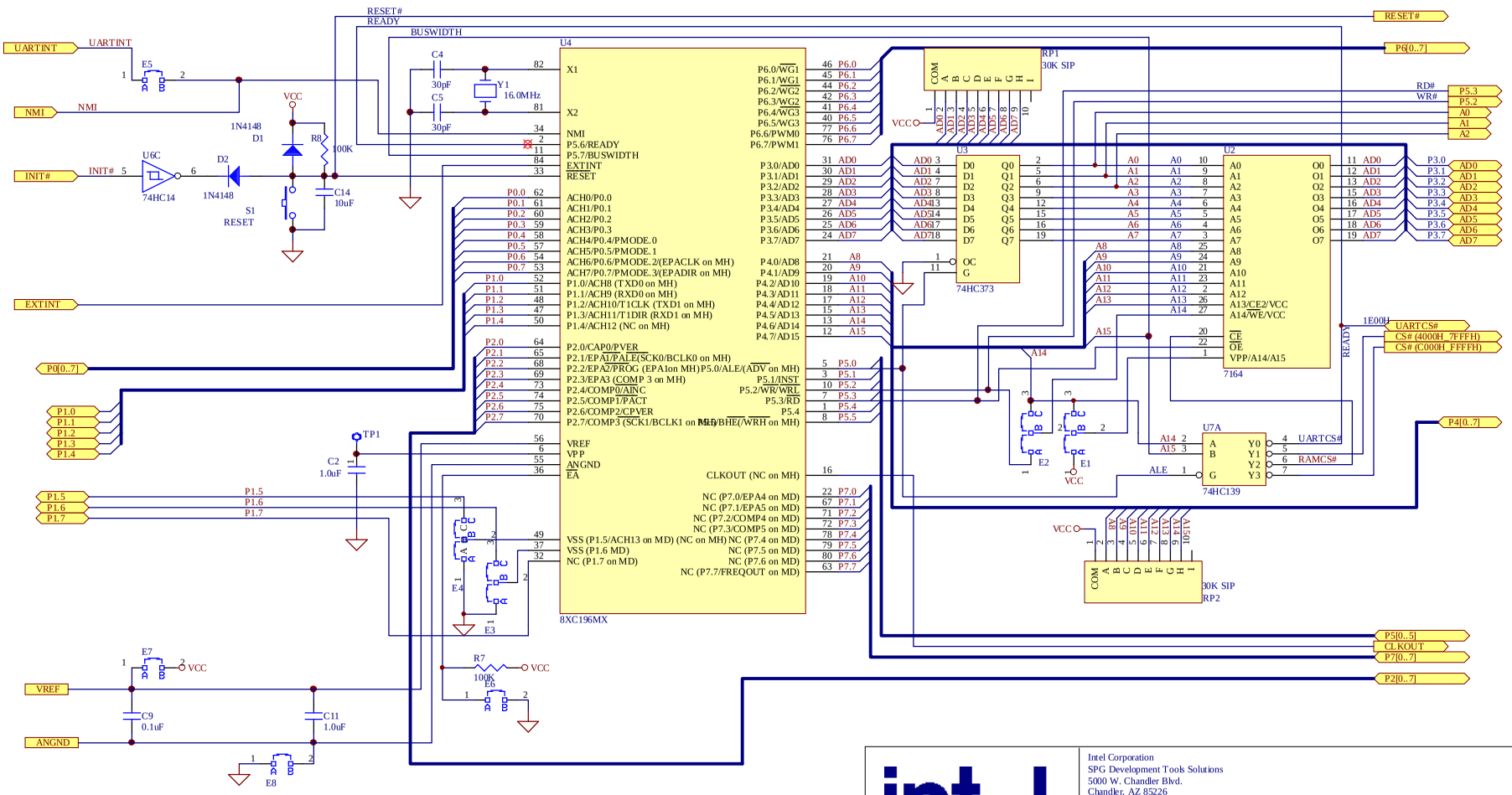
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

196MX DEMO BOARD

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\CLIENT\196MX\MX96_BUS.SCH	ERH	11-Mar-1997 10:11:03	3 of 3



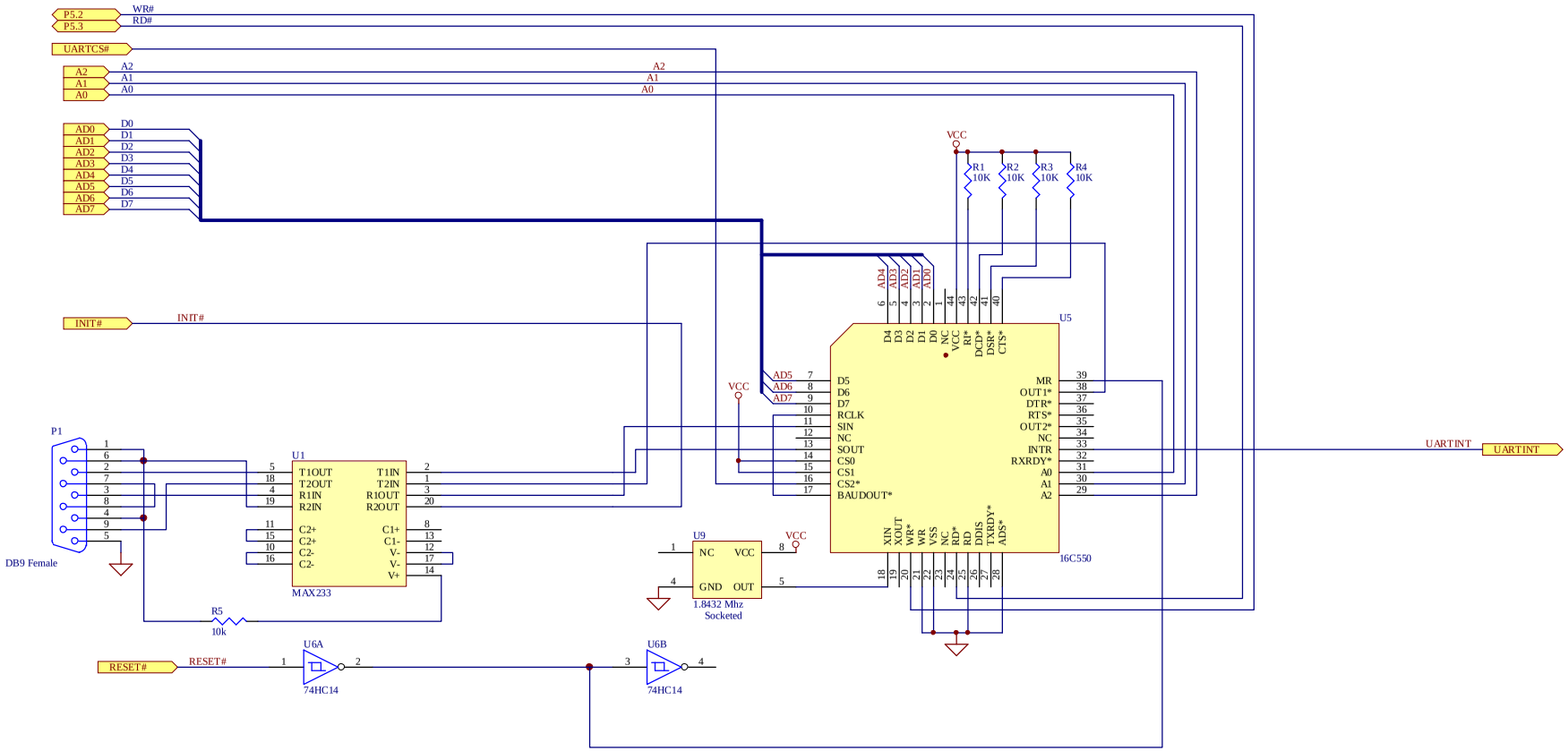
Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

196MX DEMO BOARD

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\CLIENT\196MX\MX96_CPU.SCH	ERH	11-Mar-1997 10:15:02	1 of 3



Although Intel has verified this design to be functional, Intel assumes no responsibility for any errors which may appear in the design. Intel reserves the right to modify this design without notice.

Intel Corporation
SPG Development Tools Solutions
5000 W. Chandler Blvd.
Chandler, AZ 85226

196MX DEMO BOARD

Size	FCSM No.	DWG No.	Rev
B		0001	B

File	Drawn by	Date	Sheet
C:\CLIENT\196MX\MX96_SER.SCH	ERH	11-Mar-1997 10:17:10	2 of 3